Package ‘fuzzywuzzyR’

September 11, 2021

Type Package
Title Fuzzy String Matching
Version 1.0.5
Date 2021-09-11

BugReports https://github.com/mlampros/fuzzywuzzyR/issues

URL https://github.com/mlampros/fuzzywuzzyR

Description Fuzzy string matching implementation of the 'fuzzywuzzy' <https://github.com/seatgeek/fuzzywuzzy> 'python' package. It uses the Levenshtein Distance <https://en.wikipedia.org/wiki/Levenshtein_distance> to calculate the differences between sequences.

License GPL-2

SystemRequirements Python (>= 2.4), difflib, fuzzywuzzy (> =0.15 .0 ), python-Levenshtein ( >=0.12 .0 ). Detailed installation instructions for each operating system can be found in the README file.

Depends R(>= 3.2.3)
Imports reticulate, R6
Suggests testthat, covr, knitr, rmarkdown

Encoding UTF-8

VignetteBuilder knitr
RoxygenNote 7.1.1

NeedsCompilation no

Author Lampros Mouselimis [aut, cre] (<https://orcid.org/0000-0002-8024-1546>), SeatGeek Inc [cph]

Maintainer Lampros Mouselimis <mouselimislampros@gmail.com>

Repository CRAN

Date/Publication 2021-09-11 13:50:02 UTC
R topics documented:

check_availability ........................................... 2
FuzzExtract .................................................. 2
FuzzMatcher ................................................... 7
FuzzUtils ....................................................... 13
GetCloseMatches ................................................ 16
SequenceMatcher ............................................... 17

Index 21

check_availability  This function checks if all relevant python modules are available

Description

This function checks if all relevant python modules are available

Usage

check_availability()

FuzzExtract  Fuzzy extraction from a sequence

Description

Fuzzy extraction from a sequence

Usage

# init <- FuzzExtract$new(decoding = NULL)

Details

decoding parameter is useful in case of non-ascii character strings. If this parameter is not
NULL then the force_ascii parameter (if applicable) is internally set to FALSE. Decoding applies
only to python 2 configurations, as in python 3 character strings are decoded to unicode by default.

the Extract method selects the best match of a character string vector. It returns a list with the match
and it’s score.

the ExtractBests method returns a list of the best matches for a sequence of character strings.

the ExtractWithoutOrder method returns the best match of a character string vector (in python it
returns a generator of tuples containing the match and it’s score).
the ExtractOne method finds the single best match above a score for a character string vector. This is a convenience method which returns the single best choice.

the Dedupe is a convenience method which takes a character string vector containing duplicates and uses fuzzy matching to identify and remove duplicates. Specifically, it uses the Extract method to identify duplicates that score greater than a user defined threshold. Then, it looks for the longest item in the duplicate vector since we assume this item contains the most entity information and returns that. It breaks string length ties on an alphabetical sort. Note: as the threshold DECREASES the number of duplicates that are found INCREASES. This means that the returned deduplicated list will likely be shorter. Raise the threshold for fuzzy_dedupe to be less sensitive.

Methods

FuzzExtract$new(decoding = NULL)
--------------

Extract(string = NULL, sequence_strings = NULL, processor = NULL, scorer = NULL, limit = 5L)
--------------

ExtractBests(string = NULL, sequence_strings = NULL, processor = NULL, scorer = NULL, score_cutoff = 0L, limit = 5L)
--------------

ExtractWithoutOrder(string = NULL, sequence_strings = NULL, processor = NULL, scorer = NULL, score_cutoff = 0L)
--------------

ExtractOne(string = NULL, sequence_strings = NULL, processor = NULL, scorer = NULL, score_cutoff = 0L)
--------------

Dedupe(contains_dupes = NULL, threshold = 70L, scorer = NULL)

Methods

Public methods:

- FuzzExtract$new()
- FuzzExtract$Extract()
- FuzzExtract$ExtractBests()
- FuzzExtract$ExtractWithoutOrder()
- FuzzExtract$ExtractOne()
- FuzzExtract$Dedupe()
- FuzzExtract$clone()

Method new():

Usage:
FuzzExtract$new(decoding = NULL)

Arguments:
decoding either NULL or a character string. If not NULL then the decoding parameter takes one of the standard python encodings (such as 'utf-8'). See the details and references link for more information.

**Method Extract()**:

**Usage:**

FuzzExtract$Extract(
  string = NULL,
  sequence_strings = NULL,
  processor = NULL,
  scorer = NULL,
  limit = 5L
)

**Arguments:**

- **string**: a character string.
- **sequence_strings**: a character string vector.
- **processor**: either NULL or a function of the form f(a) -> b, where a is the query or individual choice and b is the choice to be used in matching. See the examples for more details.
- **scorer**: a function for scoring matches between the query and an individual processed choice. This should be a function of the form f(query, choice) -> int. By default, FuzzMatcher.WRATIO() is used and expects both query and choice to be strings. See the examples for more details.
- **limit**: An integer value for the maximum number of elements to be returned. Defaults to 5L.

**Method ExtractBests()**:

**Usage:**

FuzzExtract$ExtractBests(
  string = NULL,
  sequence_strings = NULL,
  processor = NULL,
  scorer = NULL,
  score_cutoff = 0L,
  limit = 5L
)

**Arguments:**

- **string**: a character string.
- **sequence_strings**: a character string vector.
- **processor**: either NULL or a function of the form f(a) -> b, where a is the query or individual choice and b is the choice to be used in matching. See the examples for more details.
- **scorer**: a function for scoring matches between the query and an individual processed choice. This should be a function of the form f(query, choice) -> int. By default, FuzzMatcher.WRATIO() is used and expects both query and choice to be strings. See the examples for more details.
- **score_cutoff**: an integer value for the score threshold. No matches with a score less than this number will be returned. Defaults to 0.
- **limit**: An integer value for the maximum number of elements to be returned. Defaults to 5L.

**Method ExtractWithoutOrder()**:
Usage:
FuzzExtract$ExtractWithoutOrder(
  string = NULL,
  sequence_strings = NULL,
  processor = NULL,
  scorer = NULL,
  score_cutoff = 0L
)

Arguments:
string a character string.
sequence_strings a character string vector
processor either NULL or a function of the form f(a) -> b, where a is the query or individual choice and b is the choice to be used in matching. See the examples for more details.
scorer a function for scoring matches between the query and an individual processed choice. This should be a function of the form f(query, choice) -> int. By default, FuzzMatcher.WRATIO() is used and expects both query and choice to be strings. See the examples for more details.
score_cutoff an integer value for the score threshold. No matches with a score less than this number will be returned. Defaults to 0

Method ExtractOne():

Usage:
FuzzExtract$ExtractOne(
  string = NULL,
  sequence_strings = NULL,
  processor = NULL,
  scorer = NULL,
  score_cutoff = 0L
)

Arguments:
string a character string.
sequence_strings a character string vector
processor either NULL or a function of the form f(a) -> b, where a is the query or individual choice and b is the choice to be used in matching. See the examples for more details.
scorer a function for scoring matches between the query and an individual processed choice. This should be a function of the form f(query, choice) -> int. By default, FuzzMatcher.WRATIO() is used and expects both query and choice to be strings. See the examples for more details.
score_cutoff an integer value for the score threshold. No matches with a score less than this number will be returned. Defaults to 0

Method Dedupe():

Usage:
FuzzExtract$Dedupe(contains_dupes = NULL, threshold = 70L, scorer = NULL)

Arguments:
contains_dupes a vector of strings that we would like to dedupe
threshold the numerical value (0, 100) point at which we expect to find duplicates. Defaults
to 70 out of 100

scorer a function for scoring matches between the query and an individual processed choice.
This should be a function of the form f(query, choice) -> int. By default, FuzzMatcher.WRATIO() is used and expects both query and choice to be strings. See the examples for more details.

**Method clone()**: The objects of this class are cloneable with this method.

*Usage:*

```r
FuzzExtract$clone(deep = FALSE)
```

*Arguments:*

- `deep` Whether to make a deep clone.

**References**

- https://github.com/seatgeek/fuzzywuzzy/blob/master/fuzzywuzzy/process.py
- https://docs.python.org/3/library/codecs.html#standard-encodings

**Examples**

```r
try({
  if (reticulate::py_available(initialize = FALSE)) {
    if (check_availability()) {
      library(fuzzywuzzyR)
      word = "new york jets"
      choices = c("Atlanta Falcons", "New York Jets", "New York Giants", "Dallas Cowboys")
      duplicat = c("Frodo Baggins", 'Tom Sawyer', 'Bilbo Baggin', 'Samuel L. Jackson',
                    'F. Baggins', 'Frody Baggins', 'Bilbo Baggins')
      # processor :
      #-------------
      init_proc = FuzzUtils$new()
      PROC = init_proc$Full_process  # class process-method
      PROC1 = tolower  # base R function
      # scorer :
      #----------
      init_scor = FuzzMatcher$new()
    }
  }
})
```
FuzzMatcher

Fuzzy character string matching (ratios)

Description
Fuzzy character string matching (ratios)
Fuzzy character string matching (ratios)

Usage
# init <- FuzzMatcher$new(decoding = NULL)

Details
decoding parameter is useful in case of non-ascii character strings. If this parameter is not NULL then the force_ascii parameter (if applicable) is internally set to FALSE. Decoding applies only to python 2 configurations, as in python 3 character strings are decoded to unicode by default.

Partial_token_set_ratio method works in the following way: 1. Find all alphanumeric tokens in each string, 2. treat them as a set, 3. construct two strings of the form, <sorted_intersection><sorted_remainder>, 4. take ratios of those two strings, 5. controls for unordered partial matches (HERE partial match is TRUE)

Partial_token_sort_ratio method returns the ratio of the most similar substring as a number between 0 and 100 but sorting the token before comparing.
the \textit{Ratio} method returns a ration in form of an integer value based on a SequenceMatcher-like class, which is built on top of the Levenshtein package (https://github.com/miohtama/python-Levenshtein)

the \textit{QRATIO} method performs a quick ratio comparison between two strings. Runs full_process from utils on both strings. Short circuits if either of the strings is empty after processing.

the \textit{WRATIO} method returns a measure of the sequences’ similarity between 0 and 100, using different algorithms. Steps in the order they occur : 1. Run full_process from utils on both strings, 2. Short circuit if this makes either string empty, 3. Take the ratio of the two processed strings (fuzz.ratio), 4. Run checks to compare the length of the strings (If one of the strings is more than 1.5 times as long as the other use partial_ratio comparisons - scale partial results by 0.9 - this makes sure only full results can return 100 - If one of the strings is over 8 times as long as the other instead scale by 0.6), 5. Run the other ratio functions (if using partial ratio functions call partial_ratio, partial_token_sort_ratio and partial_token_set_ratio scale all of these by the ratio based on length otherwise call token_sort_ratio and token_set_ratio all token based comparisons are scaled by 0.95 - on top of any partial scalars) 6. Take the highest value from these results round it and return it as an integer.

the \textit{UWRATIO} method returns a measure of the sequences’ similarity between 0 and 100, using different algorithms. Same as WRatio but preserving unicode

the \textit{UQRATIO} method returns a Unicode quick ratio. It calls \textit{QRATIO} with force_ascii set to FALSE.

the \textit{Token_sort_ratio} method returns a measure of the sequences’ similarity between 0 and 100 but sorting the token before comparing

the \textit{Partial_ratio} returns the ratio of the most similar substring as a number between 0 and 100.

the \textit{Token_set_ratio} method works in the following way : 1. Find all alphanumeric tokens in each string, 2. treat them as a set, 3. construct two strings of the form, \texttt{<sorted_intersection><sorted_remainder>}, 4. take ratios of those two strings, 5. controls for unordered partial matches (HERE partial match is FALSE)

\textbf{Methods}

\begin{verbatim}
FuzzMatcher$new(decoding = NULL)
----------------------
Partial_token_set_ratio(string1 = NULL, string2 = NULL, force_ascii = TRUE, full_process = TRUE)
----------------------
Partial_token_sort_ratio(string1 = NULL, string2 = NULL, force_ascii = TRUE, full_process = TRUE)
----------------------
Ratio(string1 = NULL, string2 = NULL)
----------------------
QRATIO(string1 = NULL, string2 = NULL, force_ascii = TRUE)
----------------------
WRATIO(string1 = NULL, string2 = NULL, force_ascii = TRUE)
\end{verbatim}
FuzzMatcher

UWRATIO(string1 = NULL, string2 = NULL)
--------------
UQRATIO(string1 = NULL, string2 = NULL)
--------------
Token_sort_ratio(string1 = NULL, string2 = NULL, force_ascii = TRUE, full_process = TRUE)
--------------
Partial_ratio(string1 = NULL, string2 = NULL)
--------------
Token_set_ratio(string1 = NULL, string2 = NULL, force_ascii = TRUE, full_process = TRUE)

Methods

Public methods:
• FuzzMatcher$new()
• FuzzMatcher$Partial_token_set_ratio()
• FuzzMatcher$Partial_token_sort_ratio()
• FuzzMatcher$Ratio()
• FuzzMatcher$QRATIO()
• FuzzMatcher$WRATIO()
• FuzzMatcher$UWRATIO()
• FuzzMatcher$UQRATIO()
• FuzzMatcher$Token_sort_ratio()
• FuzzMatcher$Partial_ratio()
• FuzzMatcher$Token_set_ratio()
• FuzzMatcher$clone()

Method new():

Usage:
FuzzMatcher$new(decoding = NULL)

Arguments:
decoding  either NULL or a character string. If not NULL then the decoding parameter takes one of the standard python encodings (such as 'utf-8'). See the details and references link for more information.

Method Partial_token_set_ratio():

Usage:
FuzzMatcher$Partial_token_set_ratio(
  string1 = NULL,
  string2 = NULL,
  force_ascii = TRUE,
  full_process = TRUE
)
Arguments:
string1 a character string.
string2 a character string.
force_ascii allow only ASCII characters (force convert to ascii)
full_process either TRUE or FALSE. If TRUE then it process the string by: 1. removing all but letters and numbers, 2. trim whitespace, 3. force to lower case

Method Partial_token_sort_ratio():
Usage:
FuzzMatcher$Partial_token_sort_ratio(
  string1 = NULL,
  string2 = NULL,
  force_ascii = TRUE,
  full_process = TRUE
)
Arguments:
string1 a character string.
string2 a character string.
force_ascii allow only ASCII characters (force convert to ascii)
full_process either TRUE or FALSE. If TRUE then it process the string by: 1. removing all but letters and numbers, 2. trim whitespace, 3. force to lower case

Method Ratio():
Usage:
FuzzMatcher$Ratio(string1 = NULL, string2 = NULL)
Arguments:
string1 a character string.
string2 a character string.

Method QRATIO():
Usage:
FuzzMatcher$QRATIO(string1 = NULL, string2 = NULL, force_ascii = TRUE)
Arguments:
string1 a character string.
string2 a character string.
force_ascii allow only ASCII characters (force convert to ascii)

Method WRATIO():
Usage:
FuzzMatcher$WRATIO(string1 = NULL, string2 = NULL, force_ascii = TRUE)
Arguments:
string1 a character string.
string2 a character string.
force_ascii allow only ASCII characters (force convert to ascii)

Method UWRATIO():
Usage:
FuzzMatcher$UWRATIO(string1 = NULL, string2 = NULL)
Arguments:
string1 a character string.
string2 a character string.

Method UQRATIO():
Usage:
FuzzMatcher$UQRATIO(string1 = NULL, string2 = NULL)
Arguments:
string1 a character string.
string2 a character string.

Method Token_sort_ratio():
Usage:
FuzzMatcher$Token_sort_ratio(
    string1 = NULL,
    string2 = NULL,
    force_ascii = TRUE,
    full_process = TRUE
)
Arguments:
string1 a character string.
string2 a character string.
force_ascii allow only ASCII characters (force convert to ascii)
full_process either TRUE or FALSE. If TRUE then it process the string by: 1. removing all
but letters and numbers, 2. trim whitespace, 3. force to lower case

Method Partial_ratio():
Usage:
FuzzMatcher$Partial_ratio(string1 = NULL, string2 = NULL)
Arguments:
string1 a character string.
string2 a character string.

Method Token_set_ratio():
Usage:
FuzzMatcher$Token_set_ratio(
    string1 = NULL,
    string2 = NULL,
    force_ascii = TRUE,
    full_process = TRUE
)
Arguments:
string1 a character string.
string2 a character string.
force_ascii allow only ASCII characters (force convert to ascii)
full_process either TRUE or FALSE. If TRUE then it process the string by : 1. removing all
but letters and numbers, 2. trim whitespace, 3. force to lower case

Method clone(): The objects of this class are cloneable with this method.

Usage:
FuzzMatcher$clone(deep = FALSE)

Arguments:
deep Whether to make a deep clone.

References

Examples

try{
  if (reticulate::py_available(initialize = FALSE)) {
    if (check_availability()) {
      library(fuzzywuzzyR)
      s1 = "Atlanta Falcons"
      s2 = "New York Jets"
      init = FuzzMatcher$new()
      init$Partial_token_set_ratio(string1 = s1,
                                string2 = s2,
                                force_ascii = TRUE,
                                full_process = TRUE)
      init$Partial_token_sort_ratio(string1 = s1,
                                string2 = s2,
                                force_ascii = TRUE,
                                full_process = TRUE)
      init$Ratio(string1 = s1, string2 = s2)
      init$QRATIO(string1 = s1, string2 = s2, force_ascii = TRUE)
      init$WRATIO(string1 = s1, string2 = s2, force_ascii = TRUE)
  }
}
FuzzUtils

Utility functions

Description

Utility functions

Usage

# init <- FuzzUtils$new()

Details

the `decoding` parameter is useful in case of non-ascii character strings. If this parameter is not NULL then the `force_ascii` parameter (if applicable) is internally set to FALSE. Decoding applies only to python 2 configurations, as in python 3 character strings are decoded to unicode by default.

the `Full_process` processes a string by: 1. removing all but letters and numbers, 2. trim whitespace, 3. force to lower case and 4. if `force_ascii == TRUE`, force convert to ascii

the `INTR` method returns a correctly rounded integer

the `Make_type_consistent` method converts both objects if they aren’t either both string or unicode instances to unicode

the `Asciidammit` performs ascii dammit using the following expression `bad_chars = str('').join([chr(i) for i in range(128, 256)])`. Applies to any kind of R data type.

the `Asciionly` method returns the same result as the `Asciidammit` method but for character strings using the python `.translate()` function.

the `Validate_string` method checks that the input has length and that length is greater than 0

Some of the utils functions are used as secondary methods in the `FuzzExtract` class. See the examples of the `FuzzExtract` class for more details.
Methods

FuzzUtils$new()
--------------

Full_process(string = NULL, force_ascii = TRUE, decoding = NULL)
--------------

INTR(n = 2.0)
--------------

Make_type_consistent(string1 = NULL, string2 = NULL)
--------------

Asciidammit(input = NULL)
--------------

Asciionly(string = NULL)
--------------

Validate_string(string = NULL)

Methods

Public methods:
• FuzzUtils$new()
• FuzzUtils$Full_process()
• FuzzUtils$INTR()
• FuzzUtils$Make_type_consistent()
• FuzzUtils$Asciidammit()
• FuzzUtils$Asciionly()
• FuzzUtils$Validate_string()
• FuzzUtils$clone()

Method new():
Usage:
FuzzUtils$new()

Method Full_process():
Usage:
FuzzUtils$Full_process(string = NULL, force_ascii = TRUE, decoding = NULL)

Arguments:
string a character string.
force_ascii allow only ASCII characters (force convert to ascii)
decoding either NULL or a character string. If not NULL then the decoding parameter takes
one of the standard python encodings (such as ’utf-8’). See the details and references link
for more information (in this class it applies only to the Full_process function)

Method INTR():
**Usage:**
FuzzUtils$INTR(n = 2)

**Arguments:**
n  a float number

**Method** Make_type_consistent():

**Usage:**
FuzzUtils$Make_type_consistent(string1 = NULL, string2 = NULL)

**Arguments:**
string1  a character string.
string2  a character string.

**Method** Asciidammit():

**Usage:**
FuzzUtils$Asciidammit(input = NULL)

**Arguments:**
input  any kind of data type (applies to the Asciidammit method)

**Method** Asciionly():

**Usage:**
FuzzUtils$Asciionly(string = NULL)

**Arguments:**
string  a character string.

**Method** Validate_string():

**Usage:**
FuzzUtils$Validate_string(string = NULL)

**Arguments:**
string  a character string.

**Method** clone():  The objects of this class are cloneable with this method.

**Usage:**
FuzzUtils$clone(deep = FALSE)

**Arguments:**
deep  Whether to make a deep clone.

**References**
try({
  if (reticulate::py_available(initialize = FALSE)) {
    if (check_availability()) {
      library(fuzzywuzzyR)
      s1 = 'Frodo Baggins'
      s2 = 'Bilbo Baggin'
      init = FuzzUtils$new()
      init$Full_process(string = s1, force_ascii = TRUE)
      init$INTR(n = 2.0)
      init$Make_type_consistent(string1 = s1, string2 = s2)
      #------------------------------------
      # 'Asciidammit' with character string
      #------------------------------------
      init$Asciidammit(input = s1)
      #------------------------------------
      # 'Asciidammit' with data.frame(123) [ or any kind of data type ]
      #------------------------------------
      init$Asciidammit(input = data.frame(123))
      init$Asciionly(string = s1)
      init$Validate_string(string = s2)
    }
  }
}, silent=TRUE)

---

GetCloseMatches  Matches of character strings

**Description**

Matches of character strings

**Usage**

GetCloseMatches(string = NULL, sequence_strings = NULL, n = 3L, cutoff = 0.6)
**SequenceMatcher**

**Arguments**

- **string**: a character string.
- **sequence_strings**: a vector of character strings.
- **n**: an integer value specifying the maximum number of close matches to return; **n** must be greater than 0.
- **cutoff**: a float number in the range [0, 1], **sequence_strings** that don't score at least that similar to **string** are ignored.

**Details**

Returns a list of the best "good enough" matches. **string** is a sequence for which close matches are desired (typically a string), and **sequence_strings** is a list of sequences against which to match **string** (typically a list of strings).

**References**


**Examples**

```r
try({
  if (reticulate::py_available(initialize = FALSE)) {
    if (check_availability()) {
      library(fuzzywuzzyR)
      vec = c("Frodo Baggins", "Tom Sawyer", "Bilbo Baggin")
      str1 = "Fra Bagg"
      GetCloseMatches(string = str1, sequence_strings = vec, n = 2L, cutoff = 0.6)
    }
  }
}, silent=TRUE)
```

---

**Description**

Character string sequence matching

Character string sequence matching
Usage

```r
# init <- SequenceMatcher$new(string1 = NULL, string2 = NULL)
```

Details

the `ratio` method returns a measure of the sequences' similarity as a float in the range [0, 1]. Where $T$ is the total number of elements in both sequences, and $M$ is the number of matches, this is $2.0*M / T$. Note that this is 1.0 if the sequences are identical, and 0.0 if they have nothing in common. This is expensive to compute if getMatchingBlocks() or getOpcodes() hasn’t already been called, in which case you may want to try `quickRatio()` or `realQuickRatio()` first to get an upper bound.

the `quick_ratio` method returns an upper bound on `ratio()` relatively quickly.

the `real_quick_ratio` method returns an upper bound on `ratio()` very quickly.

the `get_matching_blocks` method returns a list of triples describing matching subsequences. Each triple is of the form [i, j, n], and means that a[i:i+n] == b[j:j+n]. The triples are monotonically increasing in i and j. The last triple is a dummy, and has the value [a.length, b.length, 0]. It is the only triple with n == 0. If [i, j, n] and [i', j', n'] are adjacent triples in the list, and the second is not the last triple in the list, then i+n != i' or j+n != j'; in other words, adjacent triples always describe non-adjacent equal blocks.

The `get_opcodes` method returns a list of 5-tuples describing how to turn a into b. Each tuple is of the form [tag, i1, i2, j1, j2]. The first tuple has i1 == j1 == 0, and remaining tuples have i1 equal to the i2 from the preceding tuple, and, likewise, j1 equal to the previous j2. The tag values are strings, with these meanings: 'replace' a[i1:i2] should be replaced by b[j1:j2]. 'delete' a[i1:i2] should be deleted. Note that j1 == j2 in this case. 'insert' b[j1:j2] should be inserted at a[i1:i1]. Note that i1 == i2 in this case. 'equal' a[i1:i2] == b[j1:j2] (the sub-sequences are equal).

Methods

```r
SequenceMatcher$new(string1 = NULL, string2 = NULL)
```

```r
-------------

ratio()
-------------

quick_ratio()
-------------

real_quick_ratio()
-------------

get_matching_blocks()
-------------

get_opcodes()
```

Methods

Public methods:

- `SequenceMatcher$new()`
- `SequenceMatcher$ratio()`
SequenceMatcher

- SequenceMatcher$quick_ratio()
- SequenceMatcher$real_quick_ratio()
- SequenceMatcher$get_matching_blocks()
- SequenceMatcher$get_opcodes()
- SequenceMatcher$clone()

Method new():

Usage:
SequenceMatcher$new(string1 = NULL, string2 = NULL)

Arguments:
string1 a character string.
string2 a character string.

Method ratio():

Usage:
SequenceMatcher$ratio()

Method quick_ratio():

Usage:
SequenceMatcher$quick_ratio()

Method real_quick_ratio():

Usage:
SequenceMatcher$real_quick_ratio()

Method get_matching_blocks():

Usage:
SequenceMatcher$get_matching_blocks()

Method get_opcodes():

Usage:
SequenceMatcher$get_opcodes()

Method clone(): The objects of this class are cloneable with this method.

Usage:
SequenceMatcher$clone(deep = FALSE)

Arguments:
depth Whether to make a deep clone.

References

Examples

```r
try({
  if (reticulate::py_available(initialize = FALSE)) {
    if (check_availability()) {
      library(fuzzywuzzyR)
      s1 = 'It was a dark and stormy night. I was all alone sitting on a red chair.'
      s2 = 'It was a murky and stormy night. I was all alone sitting on a crimson chair.'
      init = SequenceMatcher$new(string1 = s1, string2 = s2)
      init$ratio()
      init$quick_ratio()
      init$real_quick_ratio()
      init$get_matching_blocks()
      init$get_opcodes()
    }
  }
}, silent=TRUE)
```
Index

check_availability, 2
FuzzExtract, 2
FuzzMatcher, 7
FuzzUtils, 13
GetCloseMatches, 16
SequenceMatcher, 17