Package: NUSS (via r-universe)

October 27, 2024

Title Mixed N-Grams and Unigram Sequence Segmentation

Version 0.1.0

Description Segmentation of short text sequences - like hashtags -
into the separated words sequence, done with the use of
dictionary, which may be built on custom corpus of texts.
Unigram dictionary is used to find most probable sequence, and
n-grams approach is used to determine possible segmentation
given the text corpus.

License GPL (>= 3)

URL <https://github.com/theogrost/NUSS>

BugReports <https://github.com/theogrost/NUSS/issues>

Depends R (>= 3.5)

Imports dplyr, magrittr, Rcpp, stringr, text2vec, textclean, utils

Suggests testthat (>= 3.0.0)

LinkingTo BH, Rcpp

Config/testthat/edition 3

Encoding UTF-8

Language en

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.1

Repository https://theogrost.r-universe.dev

RemoteUrl https://github.com/theogrost/nuss

RemoteRef HEAD

RemoteSha 2e104423fab7ec8a0fe241f285f9107a6c7865f2

Contents

---

base_dictionary   *Base dictionary with unigrams*

Description

Data contains English unigrams, with their replacements, points and ids.

Usage

```
data(base_dictionary)
```

Format

A data.frame with four columns: to_search, to_replace, points, id.

References

Created based on Wikipedia unigrams.

Examples

```
data(base_dictionary)
```

---

igrepl   *Perform inverse regex search (C++)*

Description

This function takes character vector `patterns` with regex patterns (or fixed strings), and searches for match in the `x` string. It is inverse in the meaning, that in grepl single pattern is used against multiple strings; instead, this function takes multiple patterns to fit on a single string.

Usage

```
igrepl(patterns, x, fixed = FALSE)
```

Arguments

patterns	a character vector of regex or fixed patterns.
x	a string to search for the match.
fixed	a logical, indicating whether patterns are fixed strings.

Value

Logical vector of length as patterns with true if pattern was found.

Examples

```
igrepl(c("today","b.* fox", "jumps over", "vigorous"),
"The quick brown fox jumps over the lazy dog", FALSE)
igrepl(c("today","brown fox", "jumps over", "vigorous"),
"The quick brown fox jumps over the lazy dog", TRUE)
```

---

ngrams_dictionary   *Create n-grams dictionary*

Description

ngrams_dictionary returns the data.frame containing dictionary for ngrams_segmentation.

Usage

```
ngrams_dictionary(
  texts,
  clean = TRUE,
  ngram_min = 1,
  ngram_max = 5,
  points_filter = 1
)
```

Arguments

texts	character vector, these are the texts used to create n-grams dictionary. Case-sensitive.
clean	logical, indicating if the texts should be cleaned before creating n-grams dictionary.
ngram_min	numeric, sets the minimum number of words in creating the dictionary.
ngram_max	numeric, sets the maximum number of words in creating the dictionary.
points_filter	numeric, sets the minimal number of points (occurrences) of an n-gram to be included in the dictionary.

Value

The output always will be data.frame with 4 columns: 1) to_search, 2) to_replace, 3) id, 4) points.

Examples

```
texts <- c("this is science",
           "science is #fascinatingthing",
           "this is a scientific approach",
           "science is everywhere",
           "the beauty of science")
ngrams_dictionary(texts)
ngrams_dictionary(texts,
                  clean = FALSE)
ngrams_dictionary(texts,
                  clean = TRUE,
                  ngram_min = 2,
                  ngram_max = 2)
```

---

  ngrams_segmentation   *Segmenting sequences with n-grams.*

Description

ngrams_segmentation segments input sequence into possible segmented text based on n-grams segmentation approach.

Usage

```
ngrams_segmentation(
  sequences,
  ngrams_dictionary,
  retrieve = "most-scored",
  simplify = TRUE,
  omit_zero = TRUE,
  score_formula = "points / words.number ^ 2"
)
```

Arguments

sequences	character vector, sequence to be segmented (e.g., hashtag) or without it.
ngrams_dictionary	
	data.frame, containing ids, n-grams to search, words to use for segmentation, and their points. See details.
retrieve	character vector of length 1, with formula to calculate score.
simplify	logical, if adjacent numbers should be merged into one, and underscores removed. See simplification section.

| omit_zero | logical, if words with 0 points should be omitted from word count. See simplification section. |
| score_formula | character vector of length 1, with formula to calculate score. |

Value

The output always will be data.frame. If `retrieve='all'` is used, then the return will include all possible segmentation of the given sequence.
If `retrieve='first-shortest'` is used, the first of the shortest segmentations (with respect to the order of word's appearance in the dictionary, 1 row).
If `retrieve='most-pointed'` is used, segmentation with most total points is returned (1 row).
If `retrieve='most-scored'` is used, segmentation with the highest score calculated as $score = points/words.number^2$ (or as specified by the user).
The output is not in the input order. If needed, use lapply

ngrams_dictionary

Dictionary has to be data.frame with four named columns: 1) to_search, 2) to_replace, 3) id, 4) points.
'to_search' should be column of type character, containing n-grams to look for. Word case might be used.
'to_replace' should be column of type character, containing n-grams that should be used for creating segmentation vector, if 'to_search' matches text.
'id' should be column of type numeric, containing id of unigram.
'points' should be column of type numeric, containing number of points for the word - the higher, the better. Unigrams with 0 points might be removed from the word count with omit_zero argument.
ngrams_dictionary might be created with ngrams_dictionary.

Simplification

Two arguments are possible for simplification:

- simplify - removes spaces between numbers and removes underscores,

- omit_zero - removes ids of 0-pointed unigrams, and omits them in the word count.
  By default segmented sequence will be simplified, and numbers and underscores will be removed from word count for score computing, since they are neutral as they are necessary.

Examples

```
texts <- c("this is science",
           "science is #fascinatingthing",
           "this is a scientific approach",
           "science is everywhere",
           "the beauty of science")
ndict <- ngrams_dictionary(texts)
ngrams_segmentation("thisisscience", ndict)
ngrams_segmentation("this_is_science", ndict)
ngrams_segmentation("ThisIsScience", ndict)
```

```
ngrams_segmentation("thisisscience",
                    ndict,
                    simplify=FALSE,
                    omit_zero=FALSE)
```

---

nuss	*Mixed N-Grams and Unigram Sequence Segmentation (NUSS) function*

Description

nuss returns the data.frame containing hashtag, its segmented version, ids of dictionary words, number of words it have taken to segment the hashtag, total number of points, and computed score.

Usage

```
nuss(sequences, texts)
```

Arguments

sequences	character vector, sequence to be segmented, (e.g., hashtag) or without it. Case-insensitive.
texts	character vector, these are the texts used to create n-grams and unigram dictionary. Case-insensitive.

Details

This function is an arbitrary combination of ngrams_dictionary, unigram_dictionary, ngrams_segmentation, unigram_sequence_segmentation, created to easily segment short texts based on text corpus.

Value

The output always will be data.frame with sequences, that were
The output is not in the input order. If needed, use lapply

Examples

```
texts <- c("this is science",
           "science is #fascinatingthing",
           "this is a scientific approach",
           "science is everywhere",
           "the beauty of science")
nuss(c("thisisscience", "scienceisscience"), texts)
```

unigram_dictionary	*Create unigram dictionary*

Description

unigram_dictionary returns the data.frame containing dictionary for unigram_sequence_segmentation.

Usage

```
unigram_dictionary(texts, points_filter = 1)
```

Arguments

texts   character vector, these are the texts used to create ngrams dictionary. Case-
                    sensitive.

points_filter   numeric, sets the minimal number of points (occurrences) of an unigram to be
                    included in the dictionary.

Value

The output always will be data.frame with 4 columns: 1) to_search, 2) to_replace, 3) id, 4) points.

Examples

```
texts <- c("this is science",
           "science is #fascinatingthing",
           "this is a scientific approach",
           "science is everywhere",
           "the beauty of science")
unigram_dictionary(texts)
```

unigram_sequence_segmentation	
	Segmenting sequences with unigrams

Description

unigram_sequence_segmentation segments input sequence into possible segmented text based
on unigram sequence segmentation approach.

Usage

```
unigram_sequence_segmentation(
  sequences,
  unigram_dictionary = NUSS::base_dictionary,
  retrieve = "most-scored",
  simplify = TRUE,
  omit_zero = TRUE,
  score_formula = "points / words.number ^ 2"
)
```

Arguments

sequences	character vector, sequence to be segmented (e.g., hashtag). Case-sensitive.
unigram_dictionary	
	data.frame, containing ids, words to search, words to use for segmentation, and their points. See details.
retrieve	character vector of length 1, the type of the result data.frame to be returned: 'all', 'first-shortest', 'most-pointed' or 'most-scored'. See value section.
simplify	logical, if adjacent numbers should be merged into one, and underscores removed. See simplification section.
omit_zero	logical, if words with 0 points should be omitted from word count. See simplification section.
score_formula	character vector of length 1, with formula to calculate score.

Details

This function is not intended for long strings segmentation - 70 characters should be considered too long and may take hours to complete. 15 characters takes about 0.02s, 30 characters about 0.03s.

Value

The output always will be data.frame. If `retrieve='all'` is used, then the return will include all possible segmentation of the given sequence.
If `retrieve='first-shortest'` is used, the first of the shortest segmentations (with respect to the order of word's appearance in the dictionary, 1 row).
If `retrieve='most-pointed'` is used, segmentation with most total points is returned (1 row).
If `retrieve='most-scored'` is used, segmentation with the highest score calculated as $score = points/words.number^2$ (or as specified by the user).
The output is not in the input order. If needed, use lapply

unigram_dictionary

Dictionary has to be data.frame with four named columns: 1) to_search, 2) to_replace, 3) id, 4) points.
'to_search' should be column of type character, containing unigram to look for. Word case might be used.
'to_replace' should be column of type character, containing word that should be used for creating

segmentation vector, if 'to_search' matches text.

'id' should be column of type numeric, containing id of unigram.

'points' should be column of type numeric, containing number of points for the word - the higher, the better. Unigrams with 0 points might be removed from the word count with omit_zero argument.

Simplification

Two arguments are possible for simplification:

- simplify - removes spaces between numbers and removes underscores,

- omit_zero - removes ids of 0-pointed unigrams, and omits them in the word count.
  By default segmented sequence will be simplified, and numbers and underscores will be removed from word count for score computing, since they are neutral as they are necessary.

Examples

```
# With custom dictionary
texts <- c("this is science",
           "science is #fascinatingthing",
           "this is a scientific approach",
           "science is everywhere",
           "the beauty of science")
udict <- unigram_dictionary(texts)
unigram_sequence_segmentation('thisisscience', udict)

# With built-in dictionary (English, only lowercase)
unigram_sequence_segmentation('thisisscience')
unigram_sequence_segmentation('thisisscience2024')
unigram_sequence_segmentation('thisisscience2024', simplify=FALSE, omit_zero=FALSE)
```

Index