Package 'textrank'

January 17, 2019

Type Package

Title Summarize Text by Ranking Sentences and Finding Keywords

Version 0.3.0

Maintainer Jan Wijffels <jwijffels@bnosac.be>

Author Jan Wijffels [aut, cre, cph],
BNOSAC [cph]

Description The 'textrank' algorithm is an extension of the 'Pagerank' algorithm for text. The algorithm allows to summarize text by calculating how sentences are related to one another. This is done by looking at overlapping terminology used in sentences in order to set up links between sentences. The resulting sentence network is next plugged into the 'Pagerank' algorithm which identifies the most important sentences in your text and ranks them.
In a similar way 'textrank' can also be used to extract keywords. A word network is constructed by looking if words are following one another. On top of that network the 'Pagerank' algorithm is applied to extract relevant words after which relevant words which are following one another are combined to get keywords.
More information can be found in the paper from Mihalcea, Rada & Tarau, Paul (2004) <http://www.aclweb.org/anthology/W04-3252>.

License MPL-2.0

URL https://github.com/bnosac/textrank

Encoding UTF-8

Imports utils, data.table (>= 1.9.6), igraph, digest

Suggests textreuse, knitr, udpipe (>= 0.2)

RoxygenNote 6.0.1

VignetteBuilder knitr

NeedsCompilation no

Repository CRAN

Date/Publication 2019-01-17 22:50:03 UTC

R topics documented:

`joboffer` *The text of a job offer, annotated with the package udpipe*

Description

The text of a job offer, annotated with the package udpipe

Examples

```
data(joboffer)
str(joboffer)
unique(joboffer$sentence)
```

`summary.textrank_sentences` *Extract the most important sentences which were identified with textrank_sentences*

Description

Extract the most important sentences which were identified by `textrank_sentences`

Usage

```
## S3 method for class 'textrank_sentences'
summary(object, n = 3,
  keep.sentence.order = FALSE, ...)
```

Arguments

`object`	an object of class textrank_sentences
`n`	integer indicating to extract only the top n sentences
`keep.sentence.order`	logical indicating to keep the sentence order as provided in the original `data` argument of the `textrank_sentences` function or to order it by the pagerank score. Defaults to FALSE indicating to order by pagerank score.
`...`	not used

Value

a character vector with the top n most important sentences which were identified by textrank_sentences

See Also

textrank_sentences

`textrank_candidates_all` *Get all combinations of sentences*

Description

Get all combinations of sentences

Usage

```
textrank_candidates_all(x)
```

Arguments

x	a character vector of sentence identifiers

Value

a data.frame with 2 columns textrank_id_1 and textrank_id_2 listing up all possible combinations of x. The columns textrank_id_1 and textrank_id_2 contain identifiers of sentences given in sentence_id. This data.frame can be used as input in the textrank_sentences algorithm.

See Also

textrank_sentences

Examples

```
library(udpipe)
data(joboffer)
joboffer$textrank_id <- unique_identifier(joboffer, c("doc_id", "paragraph_id", "sentence_id"))
candidates <- textrank_candidates_all(unique(joboffer$textrank_id))
head(candidates, 50)
```

`textrank_candidates_lsh` *Use locality-sensitive hashing to get combinations of sentences which contain words which are in the same minhash bucket*

Description

This functionality is usefull if there are a lot of sentences and most of the sentences have no overlapping words in there. In order not to compute the jaccard distance among all possible combinations of sentences as is done by using `textrank_candidates_all`, we can reduce the combinations of sentences by using the Minhash algorithm. This function sets up the combinations of sentences which are in the same Minhash bucket.

Usage

```
textrank_candidates_lsh(x, sentence_id, minhashFUN, bands)
```

Arguments

`x`	a character vector of words or terms
`sentence_id`	a character vector of identifiers of sentences where the words/terms provided in `x` are part of the sentence. The length of `sentence_id` should be the same length of `x`
`minhashFUN`	a function which returns a minhash of a character vector. See the examples or look at `minhash_generator`
`bands`	integer indicating to break down the minhashes in `bands` number of bands. Mark that the number of minhash signatures should always be a multiple of the number of local sensitive hashing bands. See the example

Value

a data.frame with 2 columns textrank_id_1 and textrank_id_2 containing identifiers of sentences `sentence_id` which contained terms in the same minhash bucket. This data.frame can be used as input in the `textrank_sentences` algorithm.

See Also

`textrank_sentences`

Examples

```
library(textreuse)
library(udpipe)
lsh_probability(h = 1000, b = 500, s = 0.1) # A 10 percent Jaccard overlap will be detected well

minhash <- minhash_generator(n = 1000, seed = 123456789)
```

```
data(joboffer)
joboffer$textrank_id <- unique_identifier(joboffer, c("doc_id", "paragraph_id", "sentence_id"))
sentences <- unique(joboffer[, c("textrank_id", "sentence")])
terminology <- subset(joboffer, upos %in% c("NOUN", "ADJ"), select = c("textrank_id", "lemma"))
candidates <- textrank_candidates_lsh(x = terminology$lemma, sentence_id = terminology$textrank_id,
                                      minhashFUN = minhash, bands = 500)
head(candidates)
tr <- textrank_sentences(data = sentences, terminology = terminology,
                         textrank_candidates = candidates)
summary(tr, n = 2)
```

`textrank_jaccard` *Calculate the distance between 2 vectors based on the Jaccard distance*

Description

The jaccard distance computes the percentage of terms in the 2 vectors which are overlapping.

Usage

```
textrank_jaccard(termsa, termsb)
```

Arguments

`termsa`	a character vector of words
`termsb`	a character vector of words

Value

The Jaccard distance distance between the 2 vectors

Examples

```
sentencea <- c("I", "like", "champaign")
sentenceb <- c("I", "prefer", "choco")
textrank_jaccard(termsa = sentencea, termsb = sentenceb)
```

`textrank_keywords` *Textrank - extract relevant keywords*

Description

The textrank algorithm allows to find relevant keywords in text. Where keywords are a combination of words following each other.

In order to find relevant keywords, the textrank algorithm constructs a word network. This network is constructed by looking which words follow one another. A link is set up between two words if they follow one another, the link gets a higher weight if these 2 words occur more frequenctly next to each other in the text.
On top of the resulting network the 'Pagerank' algorithm is applied to get the importance of each word. The top 1/3 of all these words are kept and are considered relevant. After this, a keywords table is constructed by combining the relevant words together if they appear following one another in the text.

Usage

```
textrank_keywords(x, relevant = rep(TRUE, length(x)), p = 1/3,
  ngram_max = 5, sep = "-")
```

Arguments

`x`	a character vector of words.
`relevant`	a logical vector indicating if the word is relevant or not. In the standard textrank algorithm, this is normally done by doing a Parts of Speech tagging and selecting which of the words are nouns and adjectives.
`p`	percentage (between 0 and 1) of relevant words to keep. Defaults to 1/3. Can also be an integer which than indicates how many words to keep. Specify +Inf if you want to keep all words.
`ngram_max`	integer indicating to limit keywords which combine `ngram_max` combinations of words which follow one another
`sep`	character string with the separator to `paste` the subsequent relevant words together

Value

an object of class textrank_keywords which is a list with elements:

- terms: a character vector of words from the word network with the highest pagerank
- pagerank: the result of a call to `page_rank` on the word network
- keywords: the data.frame with keywords containing columns keyword, ngram, freq indicating the keywords found and the frequency of occurrence

- keywords_by_ngram: data.frame with columns keyword, ngram, freq indicating the keywords found and the frequency of occurrence at each level of ngram. The difference with keywords being that if you have a sequence of words e.g. data science consultant, then in the keywords_by_ngram you would still have the keywords data analysis and science consultant, while in the keywords list element you would only have data science consultant

See Also

page_rank

Examples

```
data(joboffer)
keywords <- textrank_keywords(joboffer$lemma,
                              relevant = joboffer$upos %in% c("NOUN", "VERB", "ADJ"))
subset(keywords$keywords, ngram > 1 & freq > 1)
keywords <- textrank_keywords(joboffer$lemma,
                              relevant = joboffer$upos %in% c("NOUN"),
                              p = 1/2, sep = " ")
subset(keywords$keywords, ngram > 1)

## plotting pagerank to see the relevance of each word
barplot(sort(keywords$pagerank$vector), horiz = TRUE,
        las = 2, cex.names = 0.5, col = "lightblue", xlab = "Pagerank")
```

---

textrank_sentences *Textrank - extract relevant sentences*

Description

The textrank algorithm is a technique to rank sentences in order of importance.

In order to find relevant sentences, the textrank algorithm needs 2 inputs: a data.frame (`data`) with sentences and a data.frame (`terminology`) containing tokens which are part of each sentence.
Based on these 2 datasets, it calculates the pairwise distance between each sentence by computing how many terms are overlapping (Jaccard distance, implemented in `textrank_jaccard`). These pairwise distances among the sentences are next passed on to Google's pagerank algorithm to identify the most relevant sentences.

If `data` contains many sentences, it makes sense not to compute all pairwise sentence distances but instead limiting the calculation of the Jaccard distance to only sentence combinations which are limited by the Minhash algorithm. This is implemented in `textrank_candidates_lsh` and an example is show below.

Usage

```
textrank_sentences(data, terminology, textrank_dist = textrank_jaccard,
  textrank_candidates = textrank_candidates_all(data$textrank_id),
  max = 1000, options_pagerank = list(directed = FALSE), ...)
```

Arguments

data	a data.frame with 1 row per sentence where the first column is an identifier of a sentence (e.g. textrank_id) and the second column is the raw sentence. See the example.
terminology	a data.frame with with one row per token indicating which token is part of each sentence. The first column in this data.frame is the identifier which corresponds to the first column of data and the second column indicates the token which is part of the sentence which will be passed on to textrank_dist. See the example.
textrank_dist	a function which calculates the distance between 2 sentences which are represented by a vectors of tokens. The first 2 arguments of the function are the tokens in sentence1 and sentence2. The function should return a numeric value of length one. The larger the value, the larger the connection between the 2 vectors indicating more strength. Defaults to the jaccard distance (textrank_jaccard), indicating the percent of common tokens.
textrank_candidates	a data.frame of candidate sentence to sentence comparisons with columns textrank_id_1 and textrank_id_2 indicating for which combination of sentences we want to compute the Jaccard distance or the distance function as provided in textrank_dist. See for example textrank_candidates_all or textrank_candidates_lsh.
max	integer indicating to reduce the number of sentence to sentence combinations to compute. In case provided, we take only this max amount of rows from textrank_candidates
options_pagerank	a list of arguments passed on to page_rank
...	arguments passed on to textrank_dist

Value

an object of class textrank_sentences which is a list with elements:

- sentences: a data.frame with columns textrank_id, sentence and textrank where the textrank is the Google Pagerank importance metric of the sentence
- sentences_dist: a data.frame with columns textrank_id_1, textrank_id_2 (the sentence id) and weight which is the result of the computed distance between the 2 sentences
- pagerank: the result of a call to page_rank

See Also

page_rank, textrank_candidates_all, textrank_candidates_lsh, textrank_jaccard

Examples

```
library(udpipe)
data(joboffer)
head(joboffer)
joboffer$textrank_id <- unique_identifier(joboffer, c("doc_id", "paragraph_id", "sentence_id"))
```

```
sentences <- unique(joboffer[, c("textrank_id", "sentence")])
cat(sentences$sentence)
terminology <- subset(joboffer, upos %in% c("NOUN", "ADJ"), select = c("textrank_id", "lemma"))
head(terminology)

## Textrank for finding the most relevant sentences
tr <- textrank_sentences(data = sentences, terminology = terminology)
summary(tr, n = 2)
summary(tr, n = 5, keep.sentence.order = TRUE)

## Not run:
## Using minhash to reduce sentence combinations - relevant if you have a lot of sentences
library(textreuse)
minhash <- minhash_generator(n = 1000, seed = 123456789)
candidates <- textrank_candidates_lsh(x = terminology$lemma, sentence_id = terminology$textrank_id,
                                      minhashFUN = minhash, bands = 500)
tr <- textrank_sentences(data = sentences, terminology = terminology,
                         textrank_candidates = candidates)
summary(tr, n = 2)

## End(Not run)
## You can also reduce the number of sentence combinations by sampling
tr <- textrank_sentences(data = sentences, terminology = terminology, max = 100)
tr
summary(tr, n = 2)
```

Index