Package 'udpipe'

July 6, 2019

Type Package

Title Tokenization, Parts of Speech Tagging, Lemmatization and Dependency Parsing with the 'UDPipe' 'NLP' Toolkit

Version 0.8.3

Maintainer Jan Wijffels <jwijffels@bnosac.be>

Description This natural language processing toolkit provides language-agnostic 'tokenization', 'parts of speech tagging', 'lemmatization' and 'dependency parsing' of raw text. Next to text parsing, the package also allows you to train annotation models based on data of 'treebanks' in 'CoNLL-U' format as provided at <http://universaldependencies.org/format.html>. The techniques are explained in detail in the paper: 'Tokenizing, POS Tagging, Lemmatizing and Parsing UD 2.0 with UDPipe', available at <doi:10.18653/v1/K17-3009>.

License MPL-2.0

URL https://bnosac.github.io/udpipe/en/index.html, https://github.com/bnosac/udpipe

Encoding UTF-8

Depends R (>= 2.10)

Imports Rcpp (>= 0.11.5), data.table (>= 1.9.6), Matrix, methods

LinkingTo Rcpp

Suggests knitr, topicmodels, lattice, parallel

SystemRequirements C++11

RoxygenNote 6.0.1

VignetteBuilder knitr

NeedsCompilation yes

Author Jan Wijffels [aut, cre, cph],
BNOSAC [cph],
Institute of Formal and Applied Linguistics, Faculty of Mathematics and Physics, Charles University in Prague, Czech Republic [cph],
Milan Straka [ctb, cph],
Jana Straková [ctb, cph]

Repository CRAN

Date/Publication 2019-07-05 22:00:03 UTC

R topics documented:

`as.data.frame.udpipe_connlu` *Convert the result of udpipe_annotate to a tidy data frame*

Description

Convert the result of `udpipe_annotate` to a tidy data frame

Usage

```
## S3 method for class 'udpipe_connlu'
as.data.frame(x, ...)
```

Arguments

`x`	an object of class `udpipe_connlu` as returned by `udpipe_annotate`
`...`	currently not used

Value

a data.frame with columns doc_id, paragraph_id, sentence_id, sentence, token_id, token, lemma, upos, xpos, feats, head_token_id, dep_rel, deps, misc

The columns paragraph_id, sentence_id are integers, the other fields are character data in UTF-8 encoding.

To get more information on these fields, visit `http://universaldependencies.org/format.html` or look at `udpipe`.

See Also

`udpipe_annotate`

Examples

```
model    <- udpipe_download_model(language = "dutch-lassysmall")

if(!model$download_failed){

ud_dutch <- udpipe_load_model(model$file_model)
txt <- c("Ik ben de weg kwijt, kunt u me zeggen waar de Lange Wapper ligt? Jazeker meneer",
         "Het gaat vooruit, het gaat verbazend goed vooruit")
x <- udpipe_annotate(ud_dutch, x = txt)
x <- as.data.frame(x)
head(x)

}

## cleanup for CRAN only - you probably want to keep your model if you have downloaded it
if(file.exists(model$file_model)) file.remove(model$file_model)
```

---

as.matrix.cooccurrence

Convert the result of cooccurrence to a sparse matrix

Description

Convert the result of cooccurrence to a sparse matrix.

Usage

```
## S3 method for class 'cooccurrence'
as.matrix(x, ...)
```

Arguments

x	an object of class cooccurrence as returned by cooccurrence
...	not used

Value

a sparse matrix with in the rows and columns the terms and in the cells how many times the cooccurrence occurred

See Also

cooccurrence

Examples

```
data(brussels_reviews_anno)
## By document, which lemma's co-occur
x <- subset(brussels_reviews_anno, xpos %in% c("NN", "JJ") & language %in% "fr")
x <- cooccurrence(x, group = "doc_id", term = "lemma")
x <- as.matrix(x)
dim(x)
x[1:3, 1:3]
```

---

as_conllu *Convert a data.frame to CONLL-U format*

Description

If you have a data.frame with annotations containing 1 row per token, you can convert it to CONLL-U format with this function. The data frame is required to have the following columns: doc_id, sentence_id, sentence, token_id, token and optionally has the following columns: lemma, upos, xpos, feats, head_token_id, dep_rel, deps, misc. Where these fields have the following meaning

- doc_id: the identifier of the document
- sentence_id: the identifier of the sentence
- sentence: the text of the sentence for which this token is part of
- token_id: Word index, integer starting at 1 for each new sentence; may be a range for multi-word tokens; may be a decimal number for empty nodes.
- token: Word form or punctuation symbol.
- lemma: Lemma or stem of word form.
- upos: Universal part-of-speech tag.
- xpos: Language-specific part-of-speech tag; underscore if not available.
- feats: List of morphological features from the universal feature inventory or from a defined language-specific extension; underscore if not available.
- head_token_id: Head of the current word, which is either a value of token_id or zero (0).
- dep_rel: Universal dependency relation to the HEAD (root iff HEAD = 0) or a defined language-specific subtype of one.
- deps: Enhanced dependency graph in the form of a list of head-deprel pairs.
- misc: Any other annotation.

The tokens in the data.frame should be ordered as they appear in the sentence.

Usage

```
as_conllu(x)
```

Arguments

x	a data.frame with columns doc_id, sentence_id, sentence, token_id, token, lemma, upos, xpos, feats, head_token_id, deprel, dep_rel, misc

Value

a character string of length 1 containing the data.frame in CONLL-U format. See the example. You can easily save this to disk for processing in other applications.

References

http://universaldependencies.org/format.html

Examples

```
file_conllu <- system.file(package = "udpipe", "dummydata", "traindata.conllu")
x <- udpipe_read_conllu(file_conllu)
str(x)
conllu <- as_conllu(x)
cat(conllu)
## Not run:
## Write it to file, making sure it is in UTF-8
cat(as_conllu(x), file = file("annotations.conllu", encoding = "UTF-8"))

## End(Not run)

## Some fields are not mandatory, they will assummed to be NA
conllu <- as_conllu(x[, c('doc_id', 'sentence_id', 'sentence',
                          'token_id', 'token', 'upos')])
cat(conllu)
```

---

as_cooccurrence *Convert a matrix to a co-occurrence data.frame*

Description

Use this function to convert the cells of a matrix to a co-occurrence data.frame containing fields term1, term2 and cooc where each row of the resulting data.frame contains the value of a cell in the matrix if the cell is not empty.

Usage

```
as_cooccurrence(x)
```

Arguments

x	a matrix or sparseMatrix

Value

a data.frame with columns term1, term2 and cooc where the data in cooc contain the content of the cells in the matrix for the combination of term1 and term2

Examples

```
data(brussels_reviews_anno)
x <- subset(brussels_reviews_anno, language == "nl")
dtm <- document_term_frequencies(x = x, document = "doc_id", term = "token")
dtm <- document_term_matrix(dtm)

correlation <- dtm_cor(dtm)
cooc <- as_cooccurrence(correlation)
head(cooc)
```

---

as_phrasemachine *Convert Parts of Speech tags to one-letter tags which can be used to identify phrases based on regular expressions*

Description

Noun phrases are of common interest when doing natural language processing. Extracting noun phrases from text can be done easily by defining a sequence of Parts of Speech tags. For example this sequence of POS tags can be seen as a noun phrase: Adjective, Noun, Preposition, Noun.
This function recodes Universal POS tags to one of the following 1-letter tags, in order to simplify writing regular expressions to find Parts of Speech sequences:

- A: adjective
- C: coordinating conjuction
- D: determiner
- M: modifier of verb
- N: noun or proper noun
- P: preposition
- O: other elements

After which identifying a simple noun phrase can be just expressed by using the following regular expression (A|N)*N(P+D*(A|N)*N)* which basically says start with adjective or noun, another noun, a preposition, determiner adjective or noun and next a noun again.

Usage

```
as_phrasemachine(x, type = c("upos", "penn-treebank"))
```

Arguments

x	a character vector of POS tags for example by using udpipe_annotate
type	either 'upos' or 'penn-treebank' indicating to recode Universal Parts of Speech tags to the counterparts as described in the description, or to recode Parts of Speech tags as known in the Penn Treebank to the counterparts as described in the description

Details

For more information on extracting phrases see http://brenocon.com/handler2016phrases.pdf

Value

the character vector x where the respective POS tags are replaced with one-letter tags

See Also

phrases

Examples

```
x <- c("PROPN", "SCONJ", "ADJ", "NOUN", "VERB", "INTJ", "DET", "VERB",
       "PROPN", "AUX", "NUM", "NUM", "X", "SCONJ", "PRON", "PUNCT", "ADP",
       "X", "PUNCT", "AUX", "PROPN", "ADP", "X", "PROPN", "ADP", "DET",
       "CCONJ", "INTJ", "NOUN", "PROPN")
as_phrasemachine(x)
```

---

as_word2vec *Convert a matrix of word vectors to word2vec format*

Description

The word2vec format provides in the first line the dimension of the word vectors and in the following lines one has the elements of the wordvector where each line covers one word or token.

The function is basically a utility function which allows one to write wordvectors created with other R packages in the well-known word2vec format which is used by udpipe_train to train the dependency parser.

Usage

```
as_word2vec(x)
```

Arguments

x	a matrix with word vectors where the rownames indicate the word or token and the number of columns of the matrix indicate the side of the word vector

Value

a character string of length 1 containing the word vectors in word2vec format which can be written to a file on disk

Examples

```
wordvectors <- matrix(rnorm(1000), nrow = 100, ncol = 10)
rownames(wordvectors) <- sprintf("word%s", seq_len(nrow(wordvectors)))
wv <- as_word2vec(wordvectors)
cat(wv)

f <- file(tempfile(fileext = ".txt"), encoding = "UTF-8")
cat(wv, file = f)
close(f)
```

---

brussels_listings *Brussels AirBnB address locations available at www.insideairbnb.com*

Description

Brussels AirBnB address locations available at www.insideairbnb.com More information: http://insideairbnb.com/get-the-data.html
Data has been converted from UTF-8 to ASCII as in `iconv(x, from = "UTF-8", to = "ASCII//TRANSLIT")` in order to be able to comply to CRAN policies.

Source

http://data.insideairbnb.com/belgium/bru/brussels/2015-10-03/visualisations/listings.csv

See Also

brussels_reviews, brussels_reviews_anno

Examples

```
data(brussels_listings)
head(brussels_listings)
```

brussels_reviews *Reviews of AirBnB customers on Brussels address locations available at www.insideairbnb.com*

Description

Reviews of AirBnB customers on Brussels address locations available at www.insideairbnb.com More information: http://insideairbnb.com/get-the-data.html. The data contains 500 reviews in Spanish, 500 reviews in French and 500 reviews in Dutch.
The data frame contains the field id (unique), listing_id which corresponds to the listing_id of the brussels_listings dataset and text fields feedback and language (identified with package cld2)
Data has been converted from UTF-8 to ASCII as in `iconv(x, from = "UTF-8", to = "ASCII//TRANSLIT")` in order to be able to comply to CRAN policies.

Source

http://data.insideairbnb.com/belgium/bru/brussels/2015-10-03/visualisations/reviews.csv

See Also

brussels_listings, brussels_reviews_anno

Examples

```
data(brussels_reviews)
str(brussels_reviews)
head(brussels_reviews)
```

brussels_reviews_anno *Reviews of the AirBnB customers which are tokenised, POS tagged and lemmatised*

Description

Reviews of the AirBnB customerswhich are tokenised, POS tagged and lemmatised. The data contains 1 row per document/token and contains the fields doc_id, language, sentence_id, token_id, token, lemma, xpos.
Data has been converted from UTF-8 to ASCII as in `iconv(x, from = "UTF-8", to = "ASCII//TRANSLIT")` in order to be able to comply to CRAN policies.

Source

http://data.insideairbnb.com/belgium/bru/brussels/2015-10-03/visualisations/reviews.csv

See Also

brussels_reviews, brussels_listings

Examples

```
## brussels_reviews_anno
data(brussels_reviews_anno)
head(brussels_reviews_anno)
sort(table(brussels_reviews_anno$xpos))

## Not run:

##
## If you want to construct a similar dataset as the
## brussels_reviews_anno dataset based on the udpipe library, do as follows
##

library(udpipe)
library(data.table)
data(brussels_reviews)

## The brussels_reviews contains comments on Airbnb sites in 3 languages: es, fr and nl
table(brussels_reviews$language)
bxl_anno <- split(brussels_reviews, brussels_reviews$language)

## Annotate the Spanish comments
m <- udpipe_download_model(language = "spanish-ancora")
m <- udpipe_load_model(file = m$file_model)
bxl_anno$es <- udpipe_annotate(object = m, x = bxl_anno$es$feedback, doc_id = bxl_anno$es$id)

## Annotate the French comments
m <- udpipe_download_model(language = "french-partut")
m <- udpipe_load_model(file = m$file_model)
bxl_anno$fr <- udpipe_annotate(object = m, x = bxl_anno$fr$feedback, doc_id = bxl_anno$fr$id)

## Annotate the Dutch comments
m <- udpipe_download_model(language = "dutch-lassysmall")
m <- udpipe_load_model(file = m$file_model)
bxl_anno$nl <- udpipe_annotate(object = m, x = bxl_anno$nl$feedback, doc_id = bxl_anno$nl$id)

brussels_reviews_anno <- lapply(bxl_anno, as.data.frame)
brussels_reviews_anno <- rbindlist(brussels_reviews_anno)
str(brussels_reviews_anno)

## End(Not run)
```

---

cbind_dependencies *Add the dependency parsing information to an annotated dataset*

Description

Annotated results of udpipe_annotate contain dependency parsing results which indicate how each word is linked to another word and the relation between these 2 words.
This information is available in the fields token_id, head_token_id and dep_rel which indicates how each token is linked to the parent. The type of relation (dep_rel) is defined at http://universaldependencies.org/u/dep/index.html. For example in the text 'The economy is weak but the outlook is bright', the term economy is linked to weak as the term economy is the nominal subject of weak.
This function adds the parent information to the annotated data.frame.

Usage

```
cbind_dependencies(x, type = c("parent", "child"))
```

Arguments

x	a data.frame or data.table as returned by as.data.frame(udpipe_annotate(...))
type	currently only possible value is 'parent', indicating to add the information of the head_token_id to the dataset

Details

Mark that the output which this function provides might possibly change in subsequent releases and is experimental.

Value

a data.frame/data.table in the same order of x where the token/lemma/upos/xpos information of the parent (head dependency) is added to the data.frame. See the examples.

Examples

```
## Not run:
udmodel <- udpipe_download_model(language = "english-ewt")
udmodel <- udpipe_load_model(file = udmodel$file_model)
x <- udpipe_annotate(udmodel,
                     x = "The economy is weak but the outlook is bright")
x <- as.data.frame(x)
x[, c("token_id", "token", "head_token_id", "dep_rel")]
x <- cbind_dependencies(x, type = "parent")
nominalsubject <- subset(x, dep_rel %in% c("nsubj"))
nominalsubject <- nominalsubject[, c("dep_rel", "token", "token_parent")]
nominalsubject

## End(Not run)
```

cbind_morphological *Add morphological features to an annotated dataset*

Description

The result of udpipe_annotate which is put into a data.frame returns a field called feats containing morphological features as defined at http://universaldependencies.org/u/feat/index.html. If there are several of these features, these are concatenated with the | symbol. This function extracts each of these morphological features separately and adds these as extra columns to the data.frame

Usage

```
cbind_morphological(x, term = "feats")
```

Arguments

x	a data.frame or data.table as returned by as.data.frame(udpipe_annotate(...))
term	the name of the field in x which contains the morphological features. Defaults to 'feats'.

Value

x in the same order with extra columns added (at least the column has_morph is added indicating if any morphological features are present and as well extra columns for each possible morphological feature in the data)

Examples

```
## Not run:
udmodel <- udpipe_download_model(language = "english-ewt")
udmodel <- udpipe_load_model(file = udmodel$file_model)
x <- udpipe_annotate(udmodel,
                     x = "The economy is weak but the outlook is bright")
x <- as.data.frame(x)
x <- cbind_morphological(x, term = "feats")

## End(Not run)

f <- system.file(package = "udpipe", "dummydata", "traindata.conllu")
x <- udpipe_read_conllu(f)
x <- cbind_morphological(x, term = "feats")
```

cooccurrence *Create a cooccurence data.frame*

Description

A cooccurence data.frame indicates how many times each term co-occurs with another term.

There are 3 types of cooccurrences:

- Looking at which words are located in the same document/sentence/paragraph.
- Looking at which words are followed by another word
- Looking at which words are in the neighbourhood of the word as in follows the word within `skipgram` number of words

The output of the function gives a cooccurrence data.frame which contains the fields term1, term2 and cooc where cooc indicates how many times term1 and term2 co-occurred. This dataset can be constructed

- based upon a data frame where you look within a group (column of the data.frame) if 2 terms occurred in that group.
- based upon a vector of words in which case we look how many times each word is followed by another word.
- based upon a vector of words in which case we look how many times each word is followed by another word or is followed by another word if we skip a number of words in between.

You can also aggregate cooccurrences if you decide to do any of these 3 by a certain group and next want to have an overall aggregate.

Usage

```
cooccurrence(x, order = TRUE, ...)

## S3 method for class 'character'
cooccurrence(x, order = TRUE, ..., relevant = rep(TRUE,
  length(x)), skipgram = 0)

## S3 method for class 'cooccurrence'
cooccurrence(x, order = TRUE, ...)

## S3 method for class 'data.frame'
cooccurrence(x, order = TRUE, ..., group, term)
```

Arguments

x	either

- a data.frame where the data.frame contains 1 row per document/term, in which case you need to provide `group` and `term` where `term` is the column containing 1 term per row and `group` indicates something like a document id or document + sentence id. This uses cooccurrence.data.frame.
- a character vector with terms where one element contains 1 term. This uses cooccurrence.character.
- an object of class `cooccurrence`. This uses cooccurrence.cooccurrence.

`order` logical indicating if we need to sort the output from high cooccurrences to low coccurrences. Defaults to TRUE.

`...` other arguments passed on to the methods

`relevant` a logical vector of the same length as `x`, indicating if the word in `x` is relevant or not. This can be used to exclude stopwords from the cooccurrence calculation or selecting only nouns and adjectives to find cooccurrences along with each other (for example based on the Parts of Speech `upos` output from `udpipe_annotate`).
Only used if calculating cooccurrences on `x` which is a character vector of words.

`skipgram` integer of length 1, indicating how far in the neighbourhood to look for words. `skipgram` is considered the maximum skip distance between words to calculate co-occurrences (where co-occurrences are of type skipgram-bigram, where a skipgram-bigram are 2 words which occur at a distance of at most `skipgram + 1` from each other).
Only used if calculating cooccurrences on `x` which is a character vector of words.

`group` character vector of columns in the data frame `x` indicating to calculate cooccurrences within these columns.
This is typically a field like document id or a sentence identifier. To be used if `x` is a data.frame.

`term` character string of a column in the data frame `x`, containing 1 term per row. To be used if `x` is a data.frame.

Value

a data.frame with columns term1, term2 and cooc indicating for the combination of term1 and term2 how many times this combination occurred

Methods (by class)

- `character`: Create a cooccurence data.frame based on a vector of terms
- `cooccurrence`: Aggregate co-occurrence statistics by summing the cooc by term/term2
- `data.frame`: Create a cooccurence data.frame based on a data.frame where you look within a document / sentence / paragraph / group if terms co-occur

Examples

```
data(brussels_reviews_anno)

## By document, which lemma's co-occur
x <- subset(brussels_reviews_anno, xpos %in% c("NN", "JJ") & language %in% "fr")
x <- cooccurrence(x, group = "doc_id", term = "lemma")
```

```
head(x)

## Which words follow each other
x <- c("A", "B", "A", "B", "c")
cooccurrence(x)

data(brussels_reviews_anno)
x <- subset(brussels_reviews_anno, language == "es")
x <- cooccurrence(x$lemma)
head(x)
x <- subset(brussels_reviews_anno, language == "es")
x <- cooccurrence(x$lemma, relevant = x$xpos %in% c("NN", "JJ"), skipgram = 4)
head(x)

## Which nouns follow each other in the same document
library(data.table)
x <- as.data.table(brussels_reviews_anno)
x <- subset(x, language == "nl" & xpos %in% c("NN"))
x <- x[, cooccurrence(lemma, order = FALSE), by = list(doc_id)]
head(x)

x_nodoc <- cooccurrence(x)
x_nodoc <- subset(x_nodoc, term1 != "appartement" & term2 != "appartement")
head(x_nodoc)
```

`document_term_frequencies` *Aggregate a data.frame to the document/term level by calculating how many times a term occurs per document*

Description

Aggregate a data.frame to the document/term level by calculating how many times a term occurs per document

Usage

```
document_term_frequencies(x, document, ...)

## S3 method for class 'data.frame'
document_term_frequencies(x, document = colnames(x)[1],
  term = colnames(x)[2], ...)

## S3 method for class 'character'
document_term_frequencies(x, document = paste("doc",
  seq_along(x), sep = ""), split = "[[:space:][:punct:][:digit:]]+", ...)
```

Arguments

x	a data.frame or data.table containing a field which can be considered as a document (defaults to the first column in x) and a field which can be considered as a term (defaults to the second column in x). If the dataset also contains a column called 'freq', this will be summed over instead of counting the number of rows occur by document/term combination. If x is a character vector containing several terms, the text will be split by the argument split before doing the agregation at the document/term level.
document	If x is a data.frame, the column in x which identifies a document. If x is a character vector then document is a vector of the same length as x where document[i] is the document id which corresponds to the text in x[i].
...	further arguments passed on to the methods
term	If x is a data.frame, the column in x which identifies a term. Defaults to the second column in x.
split	The regular expression to be used if x is a character vector. This will split the character vector x in pieces by the provides split argument. Defaults to splitting according to spaces/punctuations/digits.

Value

a data.table with columns doc_id, term, freq indicating how many times a term occurred in each document. If freq occurred in the input dataset the resulting data will have summed the freq. If freq is not in the dataset, will assume that freq is 1 for each row in the input dataset x.

Methods (by class)

- data.frame: Create a data.frame with one row per document/term combination indicating the frequency of the term in the document
- character: Create a data.frame with one row per document/term combination indicating the frequency of the term in the document

Examples

```
##
## Calculate document_term_frequencies on a data.frame
##
data(brussels_reviews_anno)
x <- document_term_frequencies(brussels_reviews_anno[, c("doc_id", "token")])
x <- document_term_frequencies(brussels_reviews_anno[, c("doc_id", "lemma")])
str(x)

brussels_reviews_anno$my_doc_id <- paste(brussels_reviews_anno$doc_id,
                                         brussels_reviews_anno$sentence_id)
x <- document_term_frequencies(brussels_reviews_anno[, c("my_doc_id", "lemma")])

##
## Calculate document_term_frequencies on a character vector
##
```

```
data(brussels_reviews)
x <- document_term_frequencies(x = brussels_reviews$feedback, document = brussels_reviews$id,
                               split = " ")
x <- document_term_frequencies(x = brussels_reviews$feedback, document = brussels_reviews$id,
                               split = "[[:space:][:punct:][:digit:]]+")

##
## document-term-frequencies on several fields to easily include bigram and trigrams
##
library(data.table)
x <- as.data.table(brussels_reviews_anno)
x <- x[, token_bigram := txt_nextgram(token, n = 2), by = list(doc_id, sentence_id)]
x <- x[, token_trigram := txt_nextgram(token, n = 3), by = list(doc_id, sentence_id)]
x <- document_term_frequencies(x = x,
                               document = "doc_id",
                               term = c("token", "token_bigram", "token_trigram"))
head(x)
```

`document_term_frequencies_statistics`

Add Term Frequency, Inverse Document Frequency and Okapi BM25 statistics to the output of document_term_frequencies

---

Description

Term frequency Inverse Document Frequency (tfidf) is calculated as the multiplication of

- Term Frequency (tf): how many times the word occurs in the document / how many words are in the document
- Inverse Document Frequency (idf): log(number of documents / number of documents where the term appears)

The Okapi BM25 statistic is calculated as the multiplication of the inverse document frequency and the weighted term frequency as defined at https://en.wikipedia.org/wiki/Okapi_BM25.

Usage

```
document_term_frequencies_statistics(x, k = 1.2, b = 0.75)
```

Arguments

x	a data.table as returned by `document_term_frequencies` containing the columns doc_id, term and freq.
k	parameter k1 of the Okapi BM25 ranking function as defined at https://en.wikipedia.org/wiki/Okapi_BM25. Defaults to 1.2.
b	parameter b of the Okapi BM25 ranking function as defined at https://en.wikipedia.org/wiki/Okapi_BM25. Defaults to 0.5.

Value

a data.table with columns doc_id, term, freq and added to that the computed statistics tf, idf, tfidf, tf_bm25 and bm25.

Examples

```
data(brussels_reviews_anno)
x <- document_term_frequencies(brussels_reviews_anno[, c("doc_id", "token")])
x <- document_term_frequencies_statistics(x)
head(x)
```

---

document_term_matrix *Create a document/term matrix from a data.frame with 1 row per document/term*

Description

Create a document/term matrix from a data.frame with 1 row per document/term as returned by document_term_frequencies

Usage

```
document_term_matrix(x, vocabulary, weight = "freq", ...)

## S3 method for class 'data.frame'
document_term_matrix(x, vocabulary, weight = "freq", ...)

## S3 method for class 'DocumentTermMatrix'
document_term_matrix(x, ...)

## S3 method for class 'TermDocumentMatrix'
document_term_matrix(x, ...)

## S3 method for class 'simple_triplet_matrix'
document_term_matrix(x, ...)
```

Arguments

x	a data.frame with columns doc_id, term and freq indicating how many times a term occurred in that specific document. This is what document_term_frequencies returns. This data.frame will be reshaped to a matrix with 1 row per doc_id, the terms will be put in the columns and the freq in the matrix cells. Note that the column name to use for freq can be set in the weight argument.
vocabulary	a character vector of terms which should be present in the document term matrix even if they did not occur in x

weight	a column of x indicating what to put in the matrix cells. Defaults to 'freq' indicating to use column freq from x to put into the matrix cells
...	further arguments currently not used

Value

an sparse object of class dgCMatrix with in the rows the documents and in the columns the terms containing the frequencies provided in x extended with terms which were not in x but were provided in vocabulary. The rownames of this resulting object contain the doc_id from x

Methods (by class)

- data.frame: Construct a document term matrix from a data.frame with columns doc_id, term, freq
- DocumentTermMatrix: Convert an object of class DocumentTermMatrix from the tm package to a sparseMatrix
- TermDocumentMatrix: Convert an object of class TermDocumentMatrix from the tm package to a sparseMatrix with the documents in the rows and the terms in the columns
- simple_triplet_matrix: Convert an object of class simple_triplet_matrix from the slam package to a sparseMatrix

See Also

sparseMatrix, document_term_frequencies

Examples

```
x <- data.frame(doc_id = c(1, 1, 2, 3, 4),
 term = c("A", "C", "Z", "X", "G"),
 freq = c(1, 5, 7, 10, 0))
document_term_matrix(x)
document_term_matrix(x, vocabulary = LETTERS)

## Example on larger dataset
data(brussels_reviews_anno)
x <- document_term_frequencies(brussels_reviews_anno[, c("doc_id", "lemma")])
dtm <- document_term_matrix(x)
dim(dtm)
x <- document_term_frequencies(brussels_reviews_anno[, c("doc_id", "lemma")])
x <- document_term_frequencies_statistics(x)
dtm <- document_term_matrix(x)
dtm <- document_term_matrix(x, weight = "freq")
dtm <- document_term_matrix(x, weight = "tf_idf")
dtm <- document_term_matrix(x, weight = "bm25")

## example showing the vocubulary argument
## allowing you to making sure terms which are not in the data are provided in the resulting dtm
allterms <- unique(x$term)
dtm <- document_term_matrix(head(x, 1000), vocabulary = allterms)
```

```
##
## Example adding bigrams/trigrams to the document term matrix
## Mark that this can also be done using ?dtm_cbind
##
library(data.table)
x <- as.data.table(brussels_reviews_anno)
x <- x[, token_bigram := txt_nextgram(token, n = 2), by = list(doc_id, sentence_id)]
x <- x[, token_trigram := txt_nextgram(token, n = 3), by = list(doc_id, sentence_id)]
x <- document_term_frequencies(x = x,
                               document = "doc_id",
                               term = c("token", "token_bigram", "token_trigram"))
dtm <- document_term_matrix(x)
```

---

dtm_bind *Combine 2 document term matrices either by rows or by columns*

Description

These 2 methods provide cbind and rbind functionality for sparse matrix objects which are returned by document_term_matrix.

In case of dtm_cbind, if the rows are not ordered in the same way in x and y, it will order them based on the rownames. If there are missing rows these will be filled with NA values.
In case of dtm_rbind, if the columns are not ordered in the same way in x and y, it will order them based on the colnames. If there are missing columns these will be filled with NA values.

Usage

```
dtm_cbind(x, y)

dtm_rbind(x, y)
```

Arguments

x	a sparse matrix such as a "dgTMatrix" object which is returned by document_term_matrix
y	a sparse matrix such as a "dgTMatrix" object which is returned by document_term_matrix

Value

a sparse matrix where either rows are put below each other in case of dtm_rbind or columns are put next to each other in case of dtm_cbind

See Also

document_term_matrix

Examples

```
data(brussels_reviews_anno)
x <- brussels_reviews_anno

## rbind
dtm1 <- document_term_frequencies(x = subset(x, doc_id %in% c("10049756", "10284782")),
                                  document = "doc_id", term = "token")
dtm1 <- document_term_matrix(dtm1)
dtm2 <- document_term_frequencies(x = subset(x, doc_id %in% c("10789408", "12285061", "35509091")),
                                  document = "doc_id", term = "token")
dtm2 <- document_term_matrix(dtm2)
m <- dtm_rbind(dtm1, dtm2)
dim(m)

## cbind
library(data.table)
x <- as.data.table(brussels_reviews_anno)
x <- x[, token_bigram := txt_nextgram(token, n = 2), by = list(doc_id, sentence_id)]
dtm1 <- document_term_frequencies(x = x, document = "doc_id", term = c("token"))
dtm1 <- document_term_matrix(dtm1)
dtm2 <- document_term_frequencies(x = x, document = "doc_id", term = c("token_bigram"))
dtm2 <- document_term_matrix(dtm2)
m <- dtm_cbind(dtm1, dtm2)
dim(m)
m <- dtm_cbind(dtm1[-c(100, 999), ], dtm2[-1000,])
dim(m)
```

---

dtm_colsums *Column sums and Row sums for document term matrices*

Description

Column sums and Row sums for document term matrices

Usage

```
dtm_colsums(dtm)

dtm_rowsums(dtm)
```

Arguments

dtm	an object returned by document_term_matrix

Value

a vector of row/column sums with corresponding names

Examples

```
x <- data.frame(
 doc_id = c(1, 1, 2, 3, 4),
 term = c("A", "C", "Z", "X", "G"),
 freq = c(1, 5, 7, 10, 0))
dtm <- document_term_matrix(x)
x <- dtm_colsums(dtm)
x
x <- dtm_rowsums(dtm)
head(x)
```

`dtm_cor` *Pearson Correlation for Sparse Matrices*

Description

Pearson Correlation for Sparse Matrices. More memory and time-efficient than `cor(as.matrix(x))`.

Usage

```
dtm_cor(x)
```

Arguments

`x`	A matrix, potentially a sparse matrix such as a "dgTMatrix" object which is returned by `document_term_matrix`

Value

a correlation matrix

See Also

`document_term_matrix`

Examples

```
x <- data.frame(
 doc_id = c(1, 1, 2, 3, 4),
 term = c("A", "C", "Z", "X", "G"),
 freq = c(1, 5, 7, 10, 0))
dtm <- document_term_matrix(x)
dtm_cor(dtm)
```

dtm_remove_lowfreq *Remove terms occurring with low frequency from a Document-Term-Matrix and documents with no terms*

Description

Remove terms occurring with low frequency from a Document-Term-Matrix and documents with no terms

Usage

```
dtm_remove_lowfreq(dtm, minfreq = 5, maxterms, remove_emptydocs = TRUE)
```

Arguments

dtm	an object returned by document_term_matrix
minfreq	integer with the minimum number of times the term should occur in order to keep the term
maxterms	integer indicating the maximum number of terms which should be kept in the dtm. The argument is optional.
remove_emptydocs	logical indicating to remove documents containing no more terms after the term removal is executed. Defaults to TRUE.

Value

a sparse Matrix as returned by sparseMatrix where terms with low occurrence are removed and documents without any terms are also removed

Examples

```
data(brussels_reviews_anno)
x <- subset(brussels_reviews_anno, xpos == "NN")
x <- x[, c("doc_id", "lemma")]
x <- document_term_frequencies(x)
dtm <- document_term_matrix(x)


## Remove terms with low frequencies and documents with no terms
x <- dtm_remove_lowfreq(dtm, minfreq = 10)
dim(x)
x <- dtm_remove_lowfreq(dtm, minfreq = 10, maxterms = 25)
dim(x)
x <- dtm_remove_lowfreq(dtm, minfreq = 10, maxterms = 25, remove_emptydocs = FALSE)
dim(x)
```

dtm_remove_sparseterms *Remove terms with high sparsity from a Document-Term-Matrix*

Description

Remove terms with high sparsity from a Document-Term-Matrix and remove documents with no terms.
Sparsity indicates in how many documents the term is not occurring.

Usage

```
dtm_remove_sparseterms(dtm, sparsity = 0.99, remove_emptydocs = TRUE)
```

Arguments

dtm	an object returned by document_term_matrix
sparsity	numeric in 0-1 range indicating the sparsity percent. Defaults to 0.99 meaning drop terms which occur in less than 1 percent of the documents.
remove_emptydocs	logical indicating to remove documents containing no more terms after the term removal is executed. Defaults to TRUE.

Value

a sparse Matrix as returned by sparseMatrix where terms with high sparsity are removed and documents without any terms are also removed

Examples

```
data(brussels_reviews_anno)
x <- subset(brussels_reviews_anno, xpos == "NN")
x <- x[, c("doc_id", "lemma")]
x <- document_term_frequencies(x)
dtm <- document_term_matrix(x)


## Remove terms with low frequencies and documents with no terms
x <- dtm_remove_sparseterms(dtm, sparsity = 0.99)
dim(x)
x <- dtm_remove_sparseterms(dtm, sparsity = 0.99, remove_emptydocs = FALSE)
dim(x)
```

dtm_remove_terms *Remove terms from a Document-Term-Matrix and keep only documents which have a least some terms*

Description

Remove terms from a Document-Term-Matrix and keep only documents which have a least some terms

Usage

```
dtm_remove_terms(dtm, terms, remove_emptydocs = TRUE)
```

Arguments

dtm	an object returned by document_term_matrix
terms	a character vector of terms which are in colnames(dtm) and which should be removed
remove_emptydocs	logical indicating to remove documents containing no more terms after the term removal is executed. Defaults to TRUE.

Value

a sparse Matrix as returned by sparseMatrix where the indicated terms are removed as well as documents with no terms whatsoever

Examples

```
data(brussels_reviews_anno)
x <- subset(brussels_reviews_anno, xpos == "NN")
x <- x[, c("doc_id", "lemma")]
x <- document_term_frequencies(x)
dtm <- document_term_matrix(x)
dim(dtm)
x <- dtm_remove_terms(dtm, terms = c("appartement", "casa", "centrum", "ciudad"))
dim(x)
x <- dtm_remove_terms(dtm, terms = c("appartement", "casa", "centrum", "ciudad"),
                      remove_emptydocs = FALSE)
dim(x)
```

dtm_remove_tfidf *Remove terms from a Document-Term-Matrix and documents with no terms based on the term frequency inverse document frequency*

Description

Remove terms from a Document-Term-Matrix and documents with no terms based on the term frequency inverse document frequency. Either giving in the maximum number of terms (argument `top`), the tfidf cutoff (argument `cutoff`) or a quantile (argument `prob`)

Usage

```
dtm_remove_tfidf(dtm, top, cutoff, prob, remove_emptydocs = TRUE)
```

Arguments

`dtm`	an object returned by `document_term_matrix`
`top`	integer with the number of terms which should be kept as defined by the highest mean tfidf
`cutoff`	numeric cutoff value to keep only terms in `dtm` where the tfidf obtained by `dtm_tfidf` is higher than this value
`prob`	numeric quantile indicating to keep only terms in `dtm` where the tfidf obtained by `dtm_tfidf` is higher than the `prob` percent quantile
`remove_emptydocs`	logical indicating to remove documents containing no more terms after the term removal is executed. Defaults to `TRUE`.

Value

a sparse Matrix as returned by `sparseMatrix` where terms with high tfidf are kept and documents without any remaining terms are removed

Examples

```
data(brussels_reviews_anno)
x <- subset(brussels_reviews_anno, xpos == "NN")
x <- x[, c("doc_id", "lemma")]
x <- document_term_frequencies(x)
dtm <- document_term_matrix(x)
dtm <- dtm_remove_lowfreq(dtm, minfreq = 10)
dim(dtm)

## Keep only terms with high tfidf
x <- dtm_remove_tfidf(dtm, top=50)
dim(x)
x <- dtm_remove_tfidf(dtm, top=50, remove_emptydocs = FALSE)
dim(x)
```

```
## Keep only terms with tfidf above 1.1
x <- dtm_remove_tfidf(dtm, cutoff=1.1)
dim(x)

## Keep only terms with tfidf above the 60 percent quantile
x <- dtm_remove_tfidf(dtm, prob=0.6)
dim(x)
```

`dtm_reverse` *Inverse operation of the document_term_matrix function*

Description

Inverse operation of the `document_term_matrix` function. Creates frequency table which contains 1 row per document/term

Usage

```
dtm_reverse(x)
```

Arguments

`x`	an object as returned by `document_term_matrix`

Value

a data.frame with columns doc_id, term and freq where freq is just the value in each cell of the `x`

See Also

`document_term_matrix`

Examples

```
x <- data.frame(
 doc_id = c(1, 1, 2, 3, 4),
 term = c("A", "C", "Z", "X", "G"),
 freq = c(1, 5, 7, 10, 0))
dtm <- document_term_matrix(x)
dtm_reverse(dtm)
```

dtm_tfidf *Term Frequency - Inverse Document Frequency calculation*

Description

Term Frequency - Inverse Document Frequency calculation. Averaged by each term.

Usage

```
dtm_tfidf(dtm)
```

Arguments

dtm	an object returned by document_term_matrix

Value

a vector with tfidf values, one for each term in the dtm matrix

Examples

```
data(brussels_reviews_anno)
x <- subset(brussels_reviews_anno, xpos == "NN")
x <- x[, c("doc_id", "lemma")]
x <- document_term_frequencies(x)
dtm <- document_term_matrix(x)

## Calculate tfidf
tfidf <- dtm_tfidf(dtm)
hist(tfidf, breaks = "scott")
head(sort(tfidf, decreasing = TRUE))
head(sort(tfidf, decreasing = FALSE))
```

keywords_collocation *Extract collocations - a sequence of terms which follow each other*

Description

Collocations are a sequence of words or terms that co-occur more often than would be expected by chance. Common collocation are adjectives + nouns, nouns followed by nouns, verbs and nouns, adverbs and adjectives, verbs and prepositional phrases or verbs and adverbs.
This function extracts relevant collocations and computes the following statistics on them which are indicators of how likely two terms are collocated compared to being independent.

- PMI (pointwise mutual information): log2(P(w1w2) / P(w1) P(w2))
- MD (mutual dependency): log2(P(w1w2)^2 / P(w1) P(w2))

- LFMD (log-frequency biased mutual dependency): MD + log2(P(w1w2))

As natural language is non random - otherwise you wouldn't understand what I'm saying, most of the combinations of terms are significant. That's why these indicators of collocation are merely used to order the collocations.

Usage

```
keywords_collocation(x, term, group, ngram_max = 2, n_min = 2, sep = " ")

collocation(x, term, group, ngram_max = 2, n_min = 2, sep = " ")
```

Arguments

x	a data.frame with one row per term where the sequence of the terms correspond to the natural order of a text. The data frame x should also contain the columns provided in term and group
term	a character vector with 1 column from x which indicates the term
group	a character vector with 1 or several columns from x which indicates for example a document id or a sentence id. Collocations will be computed within this group in order not to find collocations across sentences or documents for example.
ngram_max	integer indicating the size of the collocations. Defaults to 2, indicating to compute bigrams. If set to 3, will find collocations of bigrams and trigrams.
n_min	integer indicating the frequency of how many times a collocation should at least occur in the data in order to be returned. Defaults to 2.
sep	character string with the separator which will be used to paste together terms which are collocated. Defaults to a space: ' '.

Value

a data.frame with columns

- keyword: the terms which are combined as a collocation
- ngram: the number of terms which are combined
- left: the left term of the collocation
- right: the right term of the collocation
- freq: the number of times the collocation occurred in the data
- freq_left: the number of times the left element of the collocation occurred in the data
- freq_right: the number of times the right element of the collocation occurred in the data
- pmi: the pointwise mutual information
- md: mutual dependency
- lfmd: log-frequency biased mutual dependency

Examples

```
data(brussels_reviews_anno)
x <- subset(brussels_reviews_anno, language %in% "fr")
colloc <- keywords_collocation(x, term = "lemma", group = c("doc_id", "sentence_id"),
                               ngram_max = 3, n_min = 10)
head(colloc, 10)

## Example on finding collocations of nouns preceded by an adjective
library(data.table)
x <- as.data.table(x)
x[, xpos_previous := txt_previous(xpos, n = 1), by = list(doc_id, sentence_id)]
x[, xpos_next := txt_next(xpos, n = 1), by = list(doc_id, sentence_id)]
x <- subset(x, (xpos %in% c("NN") & xpos_previous %in% c("JJ")) |
               (xpos %in% c("JJ") & xpos_next %in% c("NN")))
colloc <- keywords_collocation(x, term = "lemma", group = c("doc_id", "sentence_id"),
                               ngram_max = 2, n_min = 2)
head(colloc)
```

---

keywords_phrases *Extract phrases - a sequence of terms which follow each other based on a sequence of Parts of Speech tags*

Description

This function allows to extract phrases, like simple noun phrases, complex noun phrases or any exact sequence of parts of speech tag patterns.
An example use case of this is to get all text where an adjective is followed by a noun or for example to get all phrases consisting of a preposition which is followed by a noun which is next followed by a verb. More complex patterns are shown in the details below.

Usage

```
keywords_phrases(x, term = x, pattern, is_regex = FALSE, sep = " ",
  ngram_max = 8, detailed = TRUE)

phrases(x, term = x, pattern, is_regex = FALSE, sep = " ",
  ngram_max = 8, detailed = TRUE)
```

Arguments

x	a character vector of Parts of Speech tags where we want to locate a relevant sequence of POS tags as defined in pattern
term	a character vector of the same length as x with the words or terms corresponding to the tags in x
pattern	In case is_regex is set to FALSE, pattern should be a character vector with a sequence of POS tags to identify in x. The length of the character vector should be bigger than 1.

	In case `is_regex` is set to TRUE, this should be a regular expressions which will be used on a concatenated version of `x` to identify the locations where these regular expression occur. See the examples below.
`is_regex`	logical indicating if `pattern` can be considered as a regular expression or if it is just a character vector of POS tags. Defaults to FALSE, indicating `pattern` is not a regular expression.
`sep`	character indicating how to collapse the phrase of terms which are found. Defaults to using a space.
`ngram_max`	an integer indicating to allow phrases to be found up to `ngram` maximum number of terms following each other. Only used if is_regex is set to TRUE. Defaults to 8.
`detailed`	logical indicating to return the exact positions where the phrase was found (set to `TRUE`) or just how many times each phrase is occurring (set to `FALSE`). Defaults to `TRUE`.

Details

Common phrases which you might be interested in and which can be supplied to `pattern` are

- Simple noun phrase: "(A|N)*N(P+D*(A|N)*N)*"
- Simple verb Phrase: "((A|N)*N(P+D*(A|N)*N)*P*(M|V)*V(M|V)*|(M|V)*V(M|V)*D*(A|N)*N(P+D*(A|N)*N)*|(M
- Noun hrase with coordination conjuction: "((A(CA)*|N)*N((P(CP)*)+(D(CD)*)*(A(CA)*|N)*N)*(C(D(CD)*)*(A(CA
- Verb phrase with coordination conjuction: "(((A(CA)*|N)*N((P(CP)*)+(D(CD)*)*(A(CA)*|N)*N)*(C(D(CD)*)*(A(C

See the examples.
Mark that this functionality is also implemented in the phrasemachine package where it is implemented using plain R code, while the implementation in this package uses a more quick Rcpp implementation for extracting these kind of regular expression like phrases.

Value

If argument `detailed` is set to `TRUE` a data.frame with columns

- keyword: the phrase which corresponds to the collapsed terms of where the pattern was found
- ngram: the length of the phrase
- pattern: the pattern which was found
- start: the starting index of `x` where the pattern was found
- end: the ending index of `x` where the pattern was found

If argument `detailed` is set to `FALSE` will return aggregate frequency statistics in a data.frame containing the columns keyword, ngram and freq (how many time it is occurring)

See Also

`as_phrasemachine`

Examples

```
data(brussels_reviews_anno, package = "udpipe")
x <- subset(brussels_reviews_anno, language %in% "fr")

## Find exactly this sequence of POS tags
np <- keywords_phrases(x$xpos, pattern = c("DT", "NN", "VB", "RB", "JJ"), sep = "-")
head(np)
np <- keywords_phrases(x$xpos, pattern = c("DT", "NN", "VB", "RB", "JJ"), term = x$token)
head(np)

## Find noun phrases with the following regular expression: (A|N)+N(P+D*(A|N)*N)*
x$phrase_tag <- as_phrasemachine(x$xpos, type = "penn-treebank")
nounphrases <- keywords_phrases(x$phrase_tag, term = x$token,
                                pattern = "(A|N)+N(P+D*(A|N)*N)*", is_regex = TRUE,
                                ngram_max = 4,
                                detailed = TRUE)
head(nounphrases, 10)
head(sort(table(nounphrases$keyword), decreasing=TRUE), 20)

## Find frequent sequences of POS tags
library(data.table)
x <- as.data.table(x)
x <- x[, pos_sequence := txt_nextgram(x = xpos, n = 3), by = list(doc_id, sentence_id)]
tail(sort(table(x$pos_sequence)))
np <- keywords_phrases(x$xpos, term = x$token, pattern = c("IN", "DT", "NN"))
head(np)
```

---

keywords_rake *Keyword identification using Rapid Automatic Keyword Extraction (RAKE)*

Description

RAKE is a basic algorithm which tries to identify keywords in text. Keywords are defined as a sequence of words following one another.
The algorithm goes as follows.

- candidate keywords are extracted by looking to a contiguous sequence of words which do not contain irrelevant words
- a score is being calculated for each word which is part of any candidate keyword, this is done by
  - among the words of the candidate keywords, the algorithm looks how many times each word is occurring and how many times it co-occurs with other words
  - each word gets a score which is the ratio of the word degree (how many times it co-occurs with other words) to the word frequency
- a RAKE score for the full candidate keyword is calculated by summing up the scores of each of the words which define the candidate keyword

The resulting keywords are returned as a data.frame together with their RAKE score.

Usage

```
keywords_rake(x, term, group, relevant = rep(TRUE, nrow(x)), ngram_max = 2,
  n_min = 2, sep = " ")
```

Arguments

x	a data.frame with one row per term as returned by as.data.frame(udpipe_annotate(...))
term	character string with a column in the data frame x, containing 1 term per row. To be used if x is a data.frame.
group	a character vector with 1 or several columns from x which indicates for example a document id or a sentence id. Keywords will be computed within this group in order not to find keywords across sentences or documents for example.
relevant	a logical vector of the same length as nrow(x), indicating if the word in the corresponding row of x is relevant or not. This can be used to exclude stopwords from the keywords calculation or for selecting only nouns and adjectives to find keywords (for example based on the Parts of Speech upos output from udpipe_annotate).
ngram_max	integer indicating the maximum number of words that there should be in each keyword
n_min	integer indicating the frequency of how many times a keywords should at least occur in the data in order to be returned. Defaults to 2.
sep	character string with the separator which will be used to paste together the terms which define the keywords. Defaults to a space: ' '.

Value

a data.frame with columns keyword, ngram and rake which is ordered from low to high rake

- keyword: the keyword
- ngram: how many terms are in the keyword
- freq: how many times did the keyword occur
- rake: the ratio of the degree to the frequency as explained in the description, summed up for all words from the keyword

References

Rose, Stuart & Engel, Dave & Cramer, Nick & Cowley, Wendy. (2010). Automatic Keyword Extraction from Individual Documents. Text Mining: Applications and Theory. 1 - 20. 10.1002/9780470689646.ch1.

Examples

```
data(brussels_reviews_anno)
x <- subset(brussels_reviews_anno, language == "nl")
keywords <- keywords_rake(x = x, term = "lemma", group = "doc_id",
                          relevant = x$xpos %in% c("NN", "JJ"))
head(keywords)
```

```
x <- subset(brussels_reviews_anno, language == "fr")
keywords <- keywords_rake(x = x, term = "lemma", group = c("doc_id", "sentence_id"),
                          relevant = x$xpos %in% c("NN", "JJ"),
                          ngram_max = 10, n_min = 2, sep = "-")
head(keywords)
```

---

paste.data.frame *Concatenate text of each group of data together*

Description

This function is similar to paste but works on a data.frame, hence paste.data.frame. It concatenates text belonging to groups of data together in one string. The function is the inverse operation of strsplit.data.frame.

Usage

```
paste.data.frame(data, term, group, collapse = " ")
```

Arguments

data	a data.frame or data.table
term	a string with a column name or a character vector of column names from data which you want to concatenate together using paste
group	a string with a column name or a character vector of column names from data indicating identifiers of groups. The text in term will be concatenated by group.
collapse	a character string that you want to use to collapse the text data together. Defaults to a single space.

Value

A data.frame with 1 row per group containing the columns from group and term where all the text in term for each group will be paste-d together, separated by the collapse argument.

See Also

strsplit.data.frame, paste

Examples

```
data(brussels_reviews_anno, package = "udpipe")
head(brussels_reviews_anno)
x <- paste.data.frame(brussels_reviews_anno,
                      term = "lemma",
                      group = c("doc_id", "sentence_id"))
str(x)
x <- paste.data.frame(brussels_reviews_anno,
```

```
                      term = c("lemma", "token"),
                      group = c("doc_id", "sentence_id"),
                      collapse = "-")
str(x)
```

---

predict.LDA_VEM *Predict method for an object of class LDA_VEM or class LDA_Gibbs*

Description

Gives either the predictions to which topic a document belongs or the term posteriors by topic indicating which terms are emitted by each topic.
If you provide in newdata a document term matrix for which a document does not contain any text and hence does not have any terms with nonzero entries, the prediction will give as topic prediction NA values (see the examples).

Usage

```
## S3 method for class 'LDA_VEM'
predict(object, newdata, type = c("topics", "terms"),
  min_posterior = -1, min_terms = 0, labels, ...)

## S3 method for class 'LDA_Gibbs'
predict(object, newdata, type = c("topics", "terms"),
  min_posterior = -1, min_terms = 0, labels, ...)
```

Arguments

object	an object of class LDA_VEM or LDA_Gibbs as returned by LDA from the topicmodels package
newdata	a document/term matrix containing data for which to make a prediction
type	either 'topic' or 'terms' for the topic predictions or the term posteriors
min_posterior	numeric in 0-1 range to output only terms emitted by each topic which have a posterior probability equal or higher than min_posterior. Only used if type is 'terms'. Provide -1 if you want to keep all values.
min_terms	integer indicating the minimum number of terms to keep in the output when type is 'terms'. Defaults to 0.
labels	a character vector of the same length as the number of topics in the topic model. Indicating how to label the topics. Only valid for type = 'topic'. Defaults to topic_prob_001 up to topic_prob_999.
...	further arguments passed on to topicmodels::posterior

Value

- in case of type = 'topic': a data.table with columns doc_id, topic (the topic number to which the document is assigned to), topic_label (the topic label) topic_prob (the posterior probability score for that topic), topic_probdiff_2nd (the probability score for that topic - the probability score for the 2nd highest topic) and the probability scores for each topic as indicated by topic_labelyourownlabel
- n case of type = 'terms': a list of data.frames with columns term and prob, giving the posterior probability that each term is emitted by the topic

See Also

posterior-methods

Examples

```
## Build document/term matrix on dutch nouns
data(brussels_reviews_anno)
data(brussels_reviews)
x <- subset(brussels_reviews_anno, language == "nl")
x <- subset(x, xpos %in% c("JJ"))
x <- x[, c("doc_id", "lemma")]
x <- document_term_frequencies(x)
dtm <- document_term_matrix(x)
dtm <- dtm_remove_lowfreq(dtm, minfreq = 10)
dtm <- dtm_remove_tfidf(dtm, top = 100)

## Fit a topicmodel using VEM
library(topicmodels)
mymodel <- LDA(x = dtm, k = 4, method = "VEM")

## Get topic terminology
terminology <- predict(mymodel, type = "terms", min_posterior = 0.05, min_terms = 3)
terminology

## Get scores alongside the topic model
dtm <- document_term_matrix(x, vocabulary = mymodel@terms)
scores <- predict(mymodel, newdata = dtm, type = "topics")
scores <- predict(mymodel, newdata = dtm, type = "topics",
                  labels = c("mylabel1", "xyz", "app-location", "newlabel"))
head(scores)
table(scores$topic)
table(scores$topic_label)
table(scores$topic, exclude = c())
table(scores$topic_label, exclude = c())

## Fit a topicmodel using Gibbs
library(topicmodels)
mymodel <- LDA(x = dtm, k = 4, method = "Gibbs")
terminology <- predict(mymodel, type = "terms", min_posterior = 0.05, min_terms = 3)
scores <- predict(mymodel, type = "topics", newdata = dtm)
```

`strsplit.data.frame` *Obtain a tokenised data frame by splitting text alongside a regular expression*

Description

Obtain a tokenised data frame by splitting text alongside a regular expression. This is the inverse operation of `paste.data.frame`.

Usage

```
strsplit.data.frame(data, term, group,
  split = "[[:space:][:punct:][:digit:]]+")
```

Arguments

`data`	a data.frame or data.table
`term`	a character with a column name from `data` which you want to split into tokens
`group`	a string with a column name or a character vector of column names from `data` indicating identifiers of groups. The text in `term` will be split into tokens by group.
`split`	a regular expression indicating how to split the `term` column. Defaults to splitting by spaces, punctuation symbols or digits. This will be passed on to `strsplit`.

Value

A tokenised data frame containing one row per token.
This data.frame has the columns from `group` and `term` where the text in column `term` will be split by the provided regular expression into tokens.

See Also

`paste.data.frame`, `strsplit`

Examples

```
data(brussels_reviews, package = "udpipe")
x <- strsplit.data.frame(brussels_reviews, term = "feedback", group = "id")
head(x)
x <- strsplit.data.frame(brussels_reviews,
                         term = c("feedback"),
                         group = c("listing_id", "language"))
head(x)
```

txt_collapse *Collapse a character vector while removing missing data.*

Description

Collapse a character vector while removing missing data.

Usage

```
txt_collapse(x, collapse = " ")
```

Arguments

x	a character vector
collapse	a character string to be used to collapse the vector. Defaults to a space: ' '.

Value

a character vector of length 1 with the content of x collapsed using paste

See Also

paste

Examples

```
txt_collapse(c(NA, "hello", "world", NA))
```

txt_contains *Check if text contains a certain pattern*

Description

Look up text which has a certain pattern. This pattern lookup is performed by executing a regular expression using grepl.

Usage

```
txt_contains(x, patterns, value = FALSE, ignore.case = TRUE, ...)
```

Arguments

x	a character vector with text
patterns	a regular expression which might be contained in x, a vector of these or a list of pattern elements where the list elements include and exclude indicate to find a pattern in x while excluding elements which have another pattern
value	logical, indicating to return the elements of x where the pattern was found or just a logical vector. Defaults to FALSE indicating to return a logical.
ignore.case	logical, if set to FALSE, the pattern matching is case sensitive and if TRUE, case is ignored during matching. Passed on to grepl
...	other parameters which can be passed on to grepl e.g. fixed/perl/useBytes

Value

a logical vector of the same length as x indicating if one of the patterns was found in x.
Or the vector of elements of x where the pattern was found in case argument value is set to TRUE

See Also

grepl

Examples

```
x <- c("The cats are eating catfood",
       "Our cat is eating the catfood",
       "the dog eats catfood, he likes it")
txt_contains(x, patterns = c("cat", "dog"))
txt_contains(x, patterns = c("cat", "dog"), value = TRUE)
txt_contains(x, patterns = c("eats"), value = TRUE)
txt_contains(x, patterns = c("^The"), ignore.case = FALSE, value = TRUE)
txt_contains(x, patterns = list(include = c("cat"), exclude = c("dog")),
             value = TRUE)
txt_contains(x, "cat") & txt_contains(x, "dog")
```

---

txt_freq *Frequency statistics of elements in a vector*

Description

Frequency statistics of elements in a vector

Usage

```
txt_freq(x, exclude = c(NA, NaN), order = TRUE)
```

Arguments

x	a vector
exclude	logical indicating to exclude values from the table. Defaults to NA and NaN.
order	logical indicating to order the resulting dataset in order of frequency. Defaults to TRUE.

Value

a data.frame with columns key, freq and freq_pct indicating the how many times each value in the vector x is occurring

Examples

```
x <- sample(LETTERS, 1000, replace = TRUE)
txt_freq(x)
x <- factor(x, levels = LETTERS)
txt_freq(x, order = FALSE)
```

---

txt_highlight *Highlight words in a character vector*

Description

Highlight words in a character vector. The words provided in terms are highlighted in the text by wrapping it around the following charater: |. So 'I like milk and sugar in my coffee' would give 'I like |milk| and sugar in my coffee' if you want to highlight the word milk

Usage

```
txt_highlight(x, terms)
```

Arguments

x	a character vector with text
terms	a vector of words to highlight which appear in x

Value

A character vector with the same length of x where the terms provided in terms are put in between || to highlight them

Examples

```
x <- "I like milk and sugar in my coffee."
txt_highlight(x, terms = "sugar")
txt_highlight(x, terms = c("milk", "my"))
```

txt_next *Get the n-th next element of a vector*

Description

Get the n-th next element of a vector

Usage

```
txt_next(x, n = 1)
```

Arguments

x	a character vector where each element is just 1 term or word
n	an integer indicating how far to look next. Defaults to 1.

Value

a character vector of the same length of x with the next element

See Also

shift

Examples

```
x <- sprintf("%s%s", LETTERS, 1:26)
txt_next(x, n = 1)

data.frame(word = x,
           word_next1 = txt_next(x, n = 1),
           word_next2 = txt_next(x, n = 2),
           stringsAsFactors = FALSE)
```

txt_nextgram *Based on a vector with a word sequence, get n-grams (looking forward)*

Description

If you have annotated your text using udpipe_annotate, your text is tokenised in a sequence of words. Based on this vector of words in sequence getting n-grams comes down to looking at the next word and the subsequent word andsoforth. These words can be pasted together to form an n-gram containing the current word, the next word up, the subsequent word, ...

Usage

```
txt_nextgram(x, n = 2, sep = " ")
```

Arguments

x	a character vector where each element is just 1 term or word
n	an integer indicating the ngram. Values of 1 will keep the x, a value of 2 will append the next term to the current term, a value of 3 will append the subsequent term and the term following that term to the current term
sep	a character element indicating how to paste the subsequent words together

Value

a character vector of the same length of x with the n-grams

See Also

paste, shift

Examples

```
x <- sprintf("%s%s", LETTERS, 1:26)
txt_nextgram(x, n = 2)

data.frame(words = x,
           bigram = txt_nextgram(x, n = 2),
           trigram = txt_nextgram(x, n = 3, sep = "-"),
           quatrogram = txt_nextgram(x, n = 4, sep = ""),
           stringsAsFactors = FALSE)

x <- c("A1", "A2", "A3", NA, "A4", "A5")
data.frame(x,
           bigram = txt_nextgram(x, n = 2, sep = "_"),
           stringsAsFactors = FALSE)
```

---

txt_previous *Get the n-th previous element of a vector*

Description

Get the n-th previous element of a vector

Usage

```
txt_previous(x, n = 1)
```

Arguments

x	a character vector where each element is just 1 term or word
n	an integer indicating how far to look back. Defaults to 1.

Value

a character vector of the same length of x with the previous element

See Also

shift

Examples

```
x <- sprintf("%s%s", LETTERS, 1:26)
txt_previous(x, n = 1)

data.frame(word = x,
           word_previous1 = txt_previous(x, n = 1),
           word_previous2 = txt_previous(x, n = 2),
           stringsAsFactors = FALSE)
```

---

txt_previousgram *Based on a vector with a word sequence, get n-grams (looking backward)*

Description

If you have annotated your text using udpipe_annotate, your text is tokenised in a sequence of words. Based on this vector of words in sequence getting n-grams comes down to looking at the previous word and the subsequent previous word andsoforth. These words can be pasted together to form an n-gram containing the second previous word, the previous word, the current word ...

Usage

```
txt_previousgram(x, n = 2, sep = " ")
```

Arguments

x	a character vector where each element is just 1 term or word
n	an integer indicating the ngram. Values of 1 will keep the x, a value of 2 will append the previous term to the current term, a value of 3 will append the second previous term term and the previous term preceding the current term to the current term
sep	a character element indicating how to paste the subsequent words together

Value

a character vector of the same length of x with the n-grams

See Also

paste, shift

Examples

```
x <- sprintf("%s%s", LETTERS, 1:26)
txt_previousgram(x, n = 2)

data.frame(words = x,
           bigram = txt_previousgram(x, n = 2),
           trigram = txt_previousgram(x, n = 3, sep = "-"),
           quatrogram = txt_previousgram(x, n = 4, sep = ""),
           stringsAsFactors = FALSE)

x <- c("A1", "A2", "A3", NA, "A4", "A5")
data.frame(x,
           bigram = txt_previousgram(x, n = 2, sep = "_"),
           stringsAsFactors = FALSE)
```

---

txt_recode *Recode text to other categories*

Description

Recode text to other categories. Values of x which correspond to from[i] will be recoded to to[i]

Usage

```
txt_recode(x, from = c(), to = c())
```

Arguments

x	a character vector
from	a character vector with values of x which you want to recode
to	a character vector with values of you want to use to recode to where you want to replace values of x which correspond to from[i] to to[i]

Value

a character vector of the same length of x where values of x which are given in from will be replaced by the corresponding element in to

See Also

match

Examples

```
x <- c("NOUN", "VERB", "NOUN", "ADV")
txt_recode(x = x,
           from = c("VERB", "ADV"),
           to = c("conjugated verb", "adverb"))
```

`txt_recode_ngram` *Recode words with compound multi-word expressions*

Description

Replace in a character vector of tokens, tokens with compound multi-word expressions. So that `c("New", "York")` will be `c("New York", NA)`.

Usage

```
txt_recode_ngram(x, compound, ngram, sep = " ")
```

Arguments

`x`	a character vector of words where you want to replace tokens with compound multi-word expressions. This is generally a character vector as returned by the token column of `as.data.frame(udpipe_annotate(txt))`
`compound`	a character vector of compound words multi-word expressions indicating terms which can be considered as one word. For example `c('New York', 'Brussels Hoofdstedelijk Gewes`
`ngram`	a integer vector of the same length as `compound` indicating how many terms there are in the specific compound multi-word expressions given by `compound`, where `compound[i]` contains `ngram[i]` words. So if `x` is `c('New York', 'Brussels Hoofdstedelijk Gewest` the ngram would be `c(2, 3)`
`sep`	separator used when the compounds were constructed by combining the words together into a compound multi-word expression. Defaults to a space: ' '.

Value

the same character vector `x` where elements in `x` will be replaced by compound multi-word expression. If will give preference to replacing with compounds with higher ngrams if these occur. See the examples.

See Also

`txt_nextgram`

Examples

```
x <- c("I", "went", "to", "New", "York", "City", "on", "holiday", ".")
y <- txt_recode_ngram(x, compound = "New York", ngram = 2, sep = " ")
data.frame(x, y)

keyw <- data.frame(keyword = c("New-York", "New-York-City"), ngram = c(2, 3))
y <- txt_recode_ngram(x, compound = keyw$keyword, ngram = keyw$ngram, sep = "-")
data.frame(x, y)

## Example replacing adjectives followed by a noun with the full compound word
data(brussels_reviews_anno)
x <- subset(brussels_reviews_anno, language == "nl")
keyw <- keywords_phrases(x$xpos, term = x$token, pattern = "JJNN",
                         is_regex = TRUE, detailed = FALSE)
head(keyw)
x$term <- txt_recode_ngram(x$token, compound = keyw$keyword, ngram = keyw$ngram)
head(x[, c("token", "term", "xpos")], 12)
```

---

txt_sample *Boilerplate function to sample one element from a vector.*

Description

Boilerplate function to sample one element from a vector.

Usage

```
txt_sample(x, na.exclude = TRUE, n = 1)
```

Arguments

x	a vector
na.exclude	logical indicating to remove NA values before taking a sample
n	integer indicating the number of items to sample from x

Value

one element sampled from the vector x

See Also

sample.int

Examples

```
txt_sample(c(NA, "hello", "world", NA))
```

`txt_sentiment` *Perform dictionary-based sentiment analysis on a tokenised data frame*

Description

This function identifies words which have a positive/negative meaning, with the addition of some basic logic regarding occurrences of amplifiers/deamplifiers and negators in the neighbourhood of the word which has a positive/negative meaning.

- If a negator is occurring in the neigbourhood, positive becomes negative or vice versa.
- If amplifiers/deamplifiers occur in the neigbourhood, these amplifier weight is added to the sentiment polarity score.

This function took inspiration from qdap::polarity but was completely re-engineered to allow to calculate similar things on a udpipe-tokenised dataset. It works on a sentence level and the negator/amplification logic can not surpass a boundary defined by the PUNCT upos parts of speech tag.

Note that if you prefer to build a supervised model to perform sentiment scoring you might be interested in looking at the ruimtehol R package https://github.com/bnosac/ruimtehol instead.

Usage

```
txt_sentiment(x, term = "lemma", polarity_terms,
  polarity_negators = character(), polarity_amplifiers = character(),
  polarity_deamplifiers = character(), amplifier_weight = 0.8,
  n_before = 4, n_after = 2, constrain = FALSE)
```

Arguments

`x`	a data.frame with the columns doc_id, paragraph_id, sentence_id, upos and the column as indicated in `term`. This is exactly what `udpipe` returns.
`term`	a character string with the name of a column of `x` where you want to apply to sentiment scoring upon
`polarity_terms`	data.frame containing terms which have positive or negative meaning. This data frame should contain the columns term and polarity where term is of type character and polarity can either be 1 or -1.
`polarity_negators`	a character vector of words which will invert the meaning of the `polarity_terms` such that -1 becomes 1 and vice versa
`polarity_amplifiers`	a character vector of words which amplify the `polarity_terms`
`polarity_deamplifiers`	a character vector of words which deamplify the `polarity_terms`

amplifier_weight
: weight which is added to the polarity score if an amplifier occurs in the neighbourhood

n_before
: integer indicating how many words before the polarity_terms word one has to look to find negators/amplifiers/deamplifiers to apply its logic

n_after
: integer indicating how many words after the polarity_terms word one has to look to find negators/amplifiers/deamplifiers to apply its logic

constrain
: logical indicating to make sure the aggregated sentiment scores is between -1 and 1

Value

a list containing

- data: the x data.frame with 2 columns added: polarity and sentiment_polarity.
  - The column polarity being just the polarity column of the polarity_terms dataset corresponding to the polarity of the term you apply the sentiment scoring
  - The columnn sentiment_polarity is the value where the amplifier/de-amplifier/negator logic is applied on.
- overall: a data.frame with one row per doc_id containing the columns doc_id, sentences, terms, sentiment_polarity, terms_positive, terms_negative, terms_negation and terms_amplification providing the aggregate sentiment_polarity score of the dataset x by doc_id as well as the terminology causing the sentiment, the number of sentences and the number of non punctuation terms in the document.

Examples

```
x <- c("I do not like whatsoever when an R package has soo many dependencies.",
       "Making other people install java is annoying,
        as it is a really painful experience in classrooms.")
## Not run:
## Do the annotation to get the data.frame needed as input to txt_sentiment
anno <- udpipe(x, "english-gum")

## End(Not run)
anno <- data.frame(doc_id = c(rep("doc1", 14), rep("doc2", 18)),
                   paragraph_id = 1,
                   sentence_id = 1,
                   lemma = c("I", "do", "not", "like", "whatsoever",
                             "when", "an", "R", "package",
                             "has", "soo", "many", "dependencies", ".",
                             "Making", "other", "people", "install",
                             "java", "is", "annoying", ",", "as",
                             "it", "is", "a", "really", "painful",
                             "experience", "in", "classrooms", "."),
                   upos = c("PRON", "AUX", "PART", "VERB", "PRON",
                            "SCONJ", "DET", "PROPN", "NOUN", "VERB",
                             "ADV", "ADJ", "NOUN", "PUNCT",
                             "VERB", "ADJ", "NOUN", "ADJ", "NOUN",
                             "AUX", "VERB", "PUNCT", "SCONJ", "PRON",
```

```
                              "AUX", "DET", "ADV", "ADJ", "NOUN",
                              "ADP", "NOUN", "PUNCT"),
                    stringsasFactors = FALSE)
scores <- txt_sentiment(x = anno,
              term = "lemma",
              polarity_terms = data.frame(term = c("annoy", "like", "painful"),
                                          polarity = c(-1, 1, -1)),
              polarity_negators = c("not", "neither"),
              polarity_amplifiers = c("pretty", "many", "really", "whatsoever"),
              polarity_deamplifiers = c("slightly", "somewhat"))
scores$overall
scores$data
scores <- txt_sentiment(x = anno,
              term = "lemma",
              polarity_terms = data.frame(term = c("annoy", "like", "painful"),
                                          polarity = c(-1, 1, -1)),
              polarity_negators = c("not", "neither"),
              polarity_amplifiers = c("pretty", "many", "really", "whatsoever"),
              polarity_deamplifiers = c("slightly", "somewhat"),
              constrain = TRUE, n_before = 4,
              n_after = 2, amplifier_weight = .8)
scores$overall
scores$data
```

`txt_show` *Boilerplate function to cat only 1 element of a character vector.*

Description

Boilerplate function to cat only 1 element of a character vector.

Usage

```
txt_show(x)
```

Arguments

x	a character vector

Value

invisible

See Also

`txt_sample`

Examples

```
txt_show(c("hello \n\n\n world", "world \n\n\n hello"))
```

`txt_tagsequence` *Identify a contiguous sequence of tags as 1 being entity*

Description

This function allows to identify contiguous sequences of text which have the same label or which follow the IOB scheme.
Named Entity Recognition or Chunking frequently follows the IOB tagging scheme where "B" means the token begins an entity, "I" means it is inside an entity, "E" means it is the end of an entity and "O" means it is not part of an entity. An example of such an annotation would be 'New', 'York', 'City', 'District' which can be tagged as 'B-LOC', 'I-LOC', 'I-LOC', 'E-LOC'.
The function looks for such sequences which start with 'B-LOC' and combines all subsequent labels of the same tagging group into 1 category. This sequence of words also gets a unique identifier such that the terms 'New', 'York', 'City', 'District' would get the same sequence identifier.

Usage

```
txt_tagsequence(x, entities)
```

Arguments

`x`	a character vector of categories in the sequence of occurring (e.g. B-LOC, I-LOC, I-PER, B-PER, O, O, B-PER)
`entities`	a list of groups, where each list element contains • start: A length 1 character string with the start element identifying a sequence start. E.g. 'B-LOC' • labels: A character vector containing all the elements which are considered being part of a same labelling sequence, including the starting element. E.g. `c('B-LOC', 'I-LOC', 'E-LOC')` The list name of the group defines the label that will be assigned to the entity. If `entities` is not provided each possible value of `x` is considered an entity. See the examples.

Value

a list with elements `entity_id` and `entity` where

- entity is a character vector of the same length as `x` containing entities , constructed by recoding `x` to the names of `names(entities)`
- entity_id is an integer vector of the same length as `x` containing unique identifiers identfying the compound label sequence such that e.g. the sequence 'B-LOC', 'I-LOC', 'I-LOC', 'E-LOC' (New York City District) would get the same `entity_id` identifier.

See the examples.

Examples

```
x <- data.frame(
  token = c("The", "chairman", "of", "the", "Nakitoma", "Corporation",
            "Donald", "Duck", "went", "skiing",
             "in", "the", "Niagara", "Falls"),
  upos = c("DET", "NOUN", "ADP", "DET", "PROPN", "PROPN",
           "PROPN", "PROPN", "VERB", "VERB",
           "ADP", "DET", "PROPN", "PROPN"),
  label = c("O", "O", "O", "O", "B-ORG", "I-ORG",
            "B-PERSON", "I-PERSON", "O", "O",
            "O", "O", "B-LOCATION", "I-LOCATION"), stringsAsFactors = FALSE)
x[, c("sequence_id", "group")] <- txt_tagsequence(x$upos)
x

##
## Define entity groups following the IOB scheme
## and combine B-LOC I-LOC I-LOC sequences as 1 group (e.g. New York City)
groups <- list(
 Location = list(start = "B-LOC", labels = c("B-LOC", "I-LOC", "E-LOC")),
 Organisation =  list(start = "B-ORG", labels = c("B-ORG", "I-ORG", "E-ORG")),
 Person = list(start = "B-PER", labels = c("B-PER", "I-PER", "E-PER")),
 Misc = list(start = "B-MISC", labels = c("B-MISC", "I-MISC", "E-MISC")))
x[, c("entity_id", "entity")] <- txt_tagsequence(x$label, groups)
x
```

---

udpipe *Tokenising, Lemmatising, Tagging and Dependency Parsing of raw text in TIF format*

Description

Tokenising, Lemmatising, Tagging and Dependency Parsing of raw text in TIF format

Usage

```
udpipe(x, object, parallel.cores = 1L, parallel.chunksize, ...)
```

Arguments

x either

- a character vector: The character vector contains the text you want to tokenize, lemmatise, tag and perform dependency parsing. The names of the character vector indicate the document identifier.
- a data.frame with columns doc_id and text: The text column contains the text you want to tokenize, lemmatise, tag and perform dependency parsing. The doc_id column indicate the document identifier.
- a list of tokens: If you have already a tokenised list of tokens and you want to enrich it by lemmatising, tagging and performing dependency parsing. The names of the list indicate the document identifier.

All text data should be in UTF-8 encoding

`object` either an object of class `udpipe_model` as returned by `udpipe_load_model`, the path to the file on disk containing the udpipe model or the language as defined by `udpipe_download_model`. If the language is provided, it will download the model using `udpipe_download_model`.

`parallel.cores` integer indicating the number of parallel cores to use to speed up the annotation. Defaults to 1 (use only 1 single thread).
If more than 1 is specified, it uses parallel::mclapply (unix) or parallel::clusterApply (windows) to run annotation in parallel. In order to do this on Windows it runs first parallel::makeCluster to set up a local socket cluster, on unix it just uses forking to parallelise the annotation.
Only set this if you have more than 1 CPU at disposal and you have large amount of data to annotate as setting up a parallel backend also takes some time plus annotations will run in chunks set by `parallel.chunksize` and for each parallel chunk the udpipe model will be loaded which takes also some time.
If `parallel.cores` is bigger than 1 and `object` is of class `udpipe_model`, it will load the corresponding file from the model again in each parallel chunk.

`parallel.chunksize` integer with the size of the chunks of text to be annotated in parallel. If not provided, defaults to the size of `x` divided by `parallel.cores`. Only used in case `parallel.cores` is bigger than 1.

`...` other elements to pass on to `udpipe_annotate` and `udpipe_download_model`

Value

a data.frame with one row per doc_id and term_id containing all the tokens in the data, the lemma, the part of speech tags, the morphological features and the dependency relationship along the tokens. The data.frame has the following fields:

- doc_id: The document identifier.
- paragraph_id: The paragraph identifier which is unique within each document.
- sentence_id: The sentence identifier which is unique within each document.
- sentence: The text of the sentence of the sentence_id.
- start: Integer index indicating in the original text where the token starts. Missing in case of tokens part of multi-word tokens which are not in the text.
- end: Integer index indicating in the original text where the token ends. Missing in case of tokens part of multi-word tokens which are not in the text.
- term_id: A row identifier which is unique within the doc_id identifier.
- token_id: Token index, integer starting at 1 for each new sentence. May be a range for multi-word tokens or a decimal number for empty nodes.
- token: The token.
- lemma: The lemma of the token.
- upos: The universal parts of speech tag of the token. See http://universaldependencies.org/format.html

- xpos: The treebank-specific parts of speech tag of the token. See http://universaldependencies.org/format.html
- feats: The morphological features of the token, separated by |. See http://universaldependencies.org/format.html
- head_token_id: Indicating what is the token_id of the head of the token, indicating to which other token in the sentence it is related. See http://universaldependencies.org/format.html
- dep_rel: The type of relation the token has with the head_token_id. See http://universaldependencies.org/format.html
- deps: Enhanced dependency graph in the form of a list of head-deprel pairs. See http://universaldependencies.org/format.html
- misc: SpacesBefore/SpacesAfter/SpacesInToken spaces before/after/inside the token. Used to reconstruct the original text. See http://ufal.mff.cuni.cz/udpipe/users-manual

The columns paragraph_id, sentence_id, term_id, start, end are integers, the other fields are character data in UTF-8 encoding.

References

https://ufal.mff.cuni.cz/udpipe, https://lindat.mff.cuni.cz/repository/xmlui/handle/11234/1-2364, http://universaldependencies.org/format.html

See Also

udpipe_load_model, as.data.frame.udpipe_connlu, udpipe_download_model, udpipe_annotate

Examples

```
model    <- udpipe_download_model(language = "dutch-lassysmall")
if(!model$download_failed){
ud_dutch <- udpipe_load_model(model)

## Tokenise, Tag and Dependency Parsing Annotation. Output is in CONLL-U format.
txt <- c("Dus. Godvermehoeren met pus in alle puisten,
  zei die schele van Van Bukburg en hij had nog gelijk ook.
  Er was toen dat liedje van tietenkonttieten kont tieten kontkontkont,
  maar dat hoefden we geenseens niet te zingen.
  Je kunt zeggen wat je wil van al die gesluierde poezenpas maar d'r kwam wel
  een vleeswarenwinkel onder te voorschijn van heb je me daar nou.

  En zo gaat het maar door.",
  "Wat die ransaap van een academici nou weer in z'n botte pan heb gehaald mag
  Joost in m'n schoen gooien, maar feit staat boven water dat het een gore
  vieze vuile ransaap is.")
names(txt) <- c("document_identifier_1", "we-like-ilya-leonard-pfeiffer")

##
## TIF tagging: tag if x is a character vector, a data frame or a token sequence
##
```

```
x <- udpipe(txt, object = ud_dutch)
x <- udpipe(data.frame(doc_id = names(txt), text = txt, stringsAsFactors = FALSE),
            object = ud_dutch)
x <- udpipe(strsplit(txt, "[[:space:][:punct:][:digit:]]+"),
            object = ud_dutch)

## You can also directly pass on the language in the call to udpipe
x <- udpipe("Dit werkt ook.", object = "dutch-lassysmall")
x <- udpipe(txt, object = "dutch-lassysmall")
x <- udpipe(data.frame(doc_id = names(txt), text = txt, stringsAsFactors = FALSE),
            object = "dutch-lassysmall")
x <- udpipe(strsplit(txt, "[[:space:][:punct:][:digit:]]+"),
            object = "dutch-lassysmall")
}

## cleanup for CRAN only - you probably want to keep your model if you have downloaded it
if(file.exists(model$file_model)) file.remove(model$file_model)
```

---

udpipe_accuracy *Evaluate the accuracy of your UDPipe model on holdout data*

Description

Get precision, recall and F1 measures on finding words / sentences / upos / xpos / features annotation as well as UAS and LAS dependency scores on holdout data in conllu format.

Usage

```
udpipe_accuracy(object, file_conllu, tokenizer = c("default", "none"),
  tagger = c("default", "none"), parser = c("default", "none"))
```

Arguments

object	an object of class udpipe_model as returned by udpipe_load_model
file_conllu	the full path to a file on disk containing holdout data in conllu format
tokenizer	a character string of length 1, which is either 'default' or 'none'
tagger	a character string of length 1, which is either 'default' or 'none'
parser	a character string of length 1, which is either 'default' or 'none'

Value

a list with 3 elements

- accuracy: A character vector with accuracy metrics.
- error: A character string with possible errors when calculating the accuracy metrics

References

https://ufal.mff.cuni.cz/udpipe, http://universaldependencies.org/format.html

See Also

udpipe_load_model

Examples

```
model <- udpipe_download_model(language = "dutch-lassysmall")
if(!model$download_failed){
ud_dutch <- udpipe_load_model(model$file_model)

file_conllu <- system.file(package = "udpipe", "dummydata", "traindata.conllu")
metrics <- udpipe_accuracy(ud_dutch, file_conllu)
metrics$accuracy
metrics <- udpipe_accuracy(ud_dutch, file_conllu,
                           tokenizer = "none", tagger = "default", parser = "default")
metrics$accuracy
metrics <- udpipe_accuracy(ud_dutch, file_conllu,
                           tokenizer = "none", tagger = "none", parser = "default")
metrics$accuracy
metrics <- udpipe_accuracy(ud_dutch, file_conllu,
                           tokenizer = "default", tagger = "none", parser = "none")
metrics$accuracy
}


## cleanup for CRAN only - you probably want to keep your model if you have downloaded it
if(file.exists(model$file_model)) file.remove(model$file_model)
```

---

udpipe_annotate *Tokenising, Lemmatising, Tagging and Dependency Parsing Annotation of raw text*

Description

Tokenising, Lemmatising, Tagging and Dependency Parsing Annotation of raw text

Usage

```
udpipe_annotate(object, x, doc_id = paste("doc", seq_along(x), sep = ""),
  tokenizer = "tokenizer", tagger = c("default", "none"),
  parser = c("default", "none"), trace = FALSE, ...)
```

Arguments

object	an object of class udpipe_model as returned by udpipe_load_model
x	a character vector in UTF-8 encoding where each element of the character vector contains text which you like to tokenize, tag and perform dependency parsing.
doc_id	an identifier of a document with the same length as x. This should be a character vector. doc_id[i] corresponds to x[i].
tokenizer	a character string of length 1, which is either 'tokenizer' (default udpipe tokenisation) or a character string with more complex tokenisation options as specified in http://ufal.mff.cuni.cz/udpipe/users-manual in which case tokenizer should be a character string where the options are put after each other using the semicolon as separation.
tagger	a character string of length 1, which is either 'default' (default udpipe POS tagging and lemmatisation) or 'none' (no POS tagging and lemmatisation needed) or a character string with more complex tagging options as specified in http://ufal.mff.cuni.cz/udpipe/users-manual in which case tagger should be a character string where the options are put after each other using the semicolon as separation.
parser	a character string of length 1, which is either 'default' (default udpipe dependency parsing) or 'none' (no dependency parsing needed) or a character string with more complex parsing options as specified in http://ufal.mff.cuni.cz/udpipe/users-manual in which case parser should be a character string where the options are put after each other using the semicolon as separation.
trace	A non-negative integer indicating to show progress on the annotation. If positive it prints out a message before each trace number of elements of x for which annotation is to be executed, allowing you to see how much of the text is already annotated. Defaults to FALSE (no progress shown).
...	currently not used

Value

a list with 3 elements

- x: The x character vector with text.
- conllu: A character vector of length 1 containing the annotated result of the annotation flow in CONLL-U format. This format is explained at http://universaldependencies.org/format.html
- error: A vector with the same length of x containing possible errors when annotating x

References

https://ufal.mff.cuni.cz/udpipe, https://lindat.mff.cuni.cz/repository/xmlui/handle/11234/1-2364, http://universaldependencies.org/format.html

See Also

udpipe_load_model, as.data.frame.udpipe_connlu

Examples

```
model    <- udpipe_download_model(language = "dutch-lassysmall")
if(!model$download_failed){
ud_dutch <- udpipe_load_model(model$file_model)

## Tokenise, Tag and Dependency Parsing Annotation. Output is in CONLL-U format.
txt <- c("Dus. Godvermehoeren met pus in alle puisten,
  zei die schele van Van Bukburg en hij had nog gelijk ook.
  Er was toen dat liedje van tietenkonttieten kont tieten kontkontkont,
  maar dat hoefden we geenseens niet te zingen.
  Je kunt zeggen wat je wil van al die gesluierde poezenpas maar d'r kwam wel
  een vleeswarenwinkel onder te voorschijn van heb je me daar nou.

  En zo gaat het maar door.",
  "Wat die ransaap van een academici nou weer in z'n botte pan heb gehaald mag
  Joost in m'n schoen gooien, maar feit staat boven water dat het een gore
  vieze vuile ransaap is.")
x <- udpipe_annotate(ud_dutch, x = txt)
cat(x$conllu)
as.data.frame(x)

## Only tokenisation
x <- udpipe_annotate(ud_dutch, x = txt, tagger = "none", parser = "none")
as.data.frame(x)

## Only tokenisation and POS tagging + lemmatisation, no dependency parsing
x <- udpipe_annotate(ud_dutch, x = txt, tagger = "default", parser = "none")
as.data.frame(x)

## Only tokenisation and dependency parsing, no POS tagging nor lemmatisation
x <- udpipe_annotate(ud_dutch, x = txt, tagger = "none", parser = "default")
as.data.frame(x)

## Provide doc_id for joining and identification purpose
x <- udpipe_annotate(ud_dutch, x = txt, doc_id = c("id1", "feedbackabc"),
                     tagger = "none", parser = "none", trace = TRUE)
as.data.frame(x)

## Mark on encodings: if your data is not in UTF-8 encoding, make sure you convert it to UTF-8
## This can be done using iconv as follows for example
udpipe_annotate(ud_dutch, x = iconv('Ik drink melk bij mijn koffie.', to = "UTF-8"))
}



## cleanup for CRAN only - you probably want to keep your model if you have downloaded it
if(file.exists(model$file_model)) file.remove(model$file_model)
```

udpipe_annotation_params *List with training options set by the UDPipe community when building models based on the Universal Dependencies data*

Description

In order to show the settings which were used by the UDPipe community when building the models made available when using udpipe_download_model, the tokenizer settings used for the different treebanks are shown below, so that you can easily use this to retrain your model directly on the corresponding UD treebank which you can download at http://universaldependencies.org/#ud-treebanks.

More information on how the models provided by the UDPipe community have been built are available at https://lindat.mff.cuni.cz/repository/xmlui/handle/11234/1-2364

References

https://lindat.mff.cuni.cz/repository/xmlui/handle/11234/1-2364

Examples

```
data(udpipe_annotation_params)
str(udpipe_annotation_params)

## settings of the tokenizer
head(udpipe_annotation_params$tokenizer)

## settings of the tagger
subset(udpipe_annotation_params$tagger, language_treebank == "nl")

## settings of the parser
udpipe_annotation_params$parser
```

udpipe_download_model *Download an UDPipe model provided by the UDPipe community for a specific language of choice*

Description

Ready-made models for 65 languages trained on 94 treebanks from http://universaldependencies.org/ are provided to you. Some of these models were provided by the UDPipe community. Other models were build using this R package. You can either download these models manually in order to use it for annotation purposes or use udpipe_download_model to download these models for a specific language of choice. You have the following options:

Usage

```
udpipe_download_model(language = c("afrikaans-afribooms",
  "ancient_greek-perseus", "ancient_greek-proiel", "arabic-padt",
  "armenian-armtdp", "basque-bdt", "belarusian-hse", "bulgarian-btb",
  "buryat-bdt", "catalan-ancora", "chinese-gsd", "classical_chinese-kyoto",
  "coptic-scriptorium", "croatian-set", "czech-cac", "czech-cltt",
  "czech-fictree", "czech-pdt", "danish-ddt", "dutch-alpino",
  "dutch-lassysmall", "english-ewt", "english-gum", "english-lines",
  "english-partut", "estonian-edt", "estonian-ewt", "finnish-ftb",
  "finnish-tdt",      "french-gsd", "french-partut", "french-sequoia",
  "french-spoken", "galician-ctg", "galician-treegal", "german-gsd",
  "gothic-proiel", "greek-gdt", "hebrew-htb", "hindi-hdtb", "hungarian-szeged",
  "indonesian-gsd", "irish-idt", "italian-isdt", "italian-partut",
 "italian-postwita", "italian-vit", "japanese-gsd", "kazakh-ktb", "korean-gsd",
  "korean-kaist", "kurmanji-mg", "latin-ittb", "latin-perseus", "latin-proiel",
  "latvian-lvtb", "lithuanian-alksnis", "lithuanian-hse", "maltese-mudt",
  "marathi-ufal",      "north_sami-giella", "norwegian-bokmaal",
  "norwegian-nynorsk", "norwegian-nynorsklia", "old_church_slavonic-proiel",
  "old_french-srcmf", "old_russian-torot", "persian-seraji", "polish-lfg",
  "polish-pdb", "polish-sz", "portuguese-bosque", "portuguese-br",
  "portuguese-gsd", "romanian-nonstandard", "romanian-rrt", "russian-gsd",
  "russian-syntagrus", "russian-taiga", "sanskrit-ufal", "serbian-set",
  "slovak-snk", "slovenian-ssj", "slovenian-sst", "spanish-ancora",
  "spanish-gsd", "swedish-lines", "swedish-talbanken",      "tamil-ttb",
  "telugu-mtg", "turkish-imst", "ukrainian-iu", "upper_sorbian-ufal",
  "urdu-udtb", "uyghur-udt", "vietnamese-vtb", "wolof-wtb"),
  model_dir = getwd(),
  udpipe_model_repo = c("jwijffels/udpipe.models.ud.2.4",
  "jwijffels/udpipe.models.ud.2.3", "jwijffels/udpipe.models.ud.2.0",
  "jwijffels/udpipe.models.conll18.baseline", "bnosac/udpipe.models.ud"),
  overwrite = TRUE, ...)
```

Arguments

language	a character string with a Universal Dependencies treebank which was used to build the model. Possible values are: afrikaans-afribooms, ancient_greek-perseus, ancient_greek-proiel, arabic-padt, armenian-armtdp, basque-bdt, belarusian-hse, bulgarian-btb, buryat-bdt, catalan-ancora, chinese-gsd, coptic-scriptorium, croatian-set, czech-cac, czech-cltt, czech-fictree, czech-pdt, danish-ddt, dutch-alpino, dutch-lassysmall, english-ewt, english-gum, english-lines, english-partut, estonian-edt, finnish-ftb, finnish-tdt, french-gsd, french-partut, french-sequoia, french-spoken, galician-ctg, galician-treegal, german-gsd, gothic-proiel, greek-gdt, hebrew-htb, hindi-hdtb, hungarian-szeged, indonesian-gsd, irish-idt, italian-isdt, italian-partut, italian-postwita, japanese-gsd, kazakh-ktb, korean-gsd, korean-kaist, kurmanji-mg, latin-ittb, latin-perseus, latin-proiel, latvian-lvtb, lithuanian-hse, maltese-mudt, marathi-ufal, north_sami-giella, norwegian-bokmaal, norwegian-nynorsk, norwegian-nynorsklia, old_church_slavonic-proiel, old_french-srcmf, persian-seraji, polish-lfg, polish-sz, portuguese-bosque,

portuguese-br, portuguese-gsd, romanian-nonstandard, romanian-rrt, russian-gsd, russian-syntagrus, russian-taiga, sanskrit-ufal, serbian-set, slovak-snk, slovenian-ssj, slovenian-sst, spanish-ancora, spanish-gsd, swedish-lines, swedish-talbanken, tamil-ttb, telugu-mtg, turkish-imst, ukrainian-iu, upper_sorbian-ufal, urdu-udtb, uyghur-udt, vietnamese-vtb

Each language should have a treebank extension (e.g. english-ewt, russian-syntagrus, dutch-alpino, ...). If you do not provide a treebank extension (e.g. only english, russian, dutch), the function will use the default treebank of that language as was used in Universal Dependencies up to version 2.1.

`model_dir` a path where the model will be downloaded to. Defaults to the current working directory

`udpipe_model_repo`
location where the models will be downloaded from. Either 'jwijffels/udpipe.models.ud.2.4', 'jwijffels/udpipe.models.ud.2.3', 'jwijffels/udpipe.models.ud.2.0', 'jwijffels/udpipe.models.conll18.basel or 'bnosac/udpipe.models.ud'.
Defaults to 'jwijffels/udpipe.models.ud.2.4'.

- 'jwijffels/udpipe.models.ud.2.4' contains models released under the CC-BY-NC-SA license constructed on Universal Dependencies 2.4 data
- 'jwijffels/udpipe.models.ud.2.3' contains models released under the CC-BY-NC-SA license constructed on Universal Dependencies 2.3 data
- 'jwijffels/udpipe.models.ud.2.0' contains models released under the CC-BY-NC-SA license constructed on Universal Dependencies 2.0 data
- 'jwijffels/udpipe.models.conll18.baseline' contains models released under the CC-BY-NC-SA license constructed on Universal Dependencies 2.2 data for the 2018 conll shared task
- 'bnosac/udpipe.models.ud' contains models mainly released under the CC-BY-SA license constructed on Universal Dependencies 2.1 data, and some models released under the GPL-3 and LGPL-LR license

See the Details section for further information on which languages are available in each of these repositories.

`overwrite` logical indicating to overwrite the file if the file was already downloaded. Defaults to `TRUE` indicating it will download the model and overwrite the file if the file already existed. If set to `FALSE`, the model will only be downloaded if it does not exist on disk yet in the `model_dir` folder.

`...` currently not used

Details

The function allows you to download the following language models based on your setting of argument `udpipe_model_repo`:

- 'jwijffels/udpipe.models.ud.2.4': https://github.com/jwijffels/udpipe.models.ud.2.4
  - UDPipe models constructed on data from Universal Dependencies 2.4

  - languages-treebanks: afrikaans-afribooms, ancient_greek-perseus, ancient_greek-proiel, arabic-padt, armenian-armtdp, basque-bdt, belarusian-hse, bulgarian-btb, catalan-ancora, chinese-gsd, classical_chinese-kyoto, coptic-scriptorium, croatian-set, czech-cac, czech-cltt, czech-fictree, czech-pdt, danish-ddt, dutch-alpino, dutch-lassysmall, english-ewt, english-gum, english-lines, english-partut, estonian-edt, estonian-ewt, finnish-ftb, finnish-tdt, french-gsd, french-partut, french-sequoia, french-spoken, galician-ctg, galician-treegal, german-gsd, gothic-proiel, greek-gdt, hebrew-htb, hindi-hdtb, hungarian-szeged, indonesian-gsd, irish-idt, italian-isdt, italian-partut, italian-postwita, italian-vit, japanese-gsd, korean-gsd, korean-kaist, latin-ittb, latin-perseus, latin-proiel, latvian-lvtb, lithuanian-alksnis, lithuanian-hse, maltese-mudt, marathi-ufal, north_sami-giella, norwegian-bokmaal, norwegian-nynorsk, norwegian-nynorsklia, old_church_slavonic-proiel, old_french-srcmf, old_russian-torot, persian-seraji, polish-lfg, polish-pdb, portuguese-bosque, portuguese-gsd, romanian-nonstandard, romanian-rrt, russian-gsd, russian-syntagrus, russian-taiga, serbian-set, slovak-snk, slovenian-ssj, slovenian-sst, spanish-ancora, spanish-gsd, swedish-lines, swedish-talbanken, tamil-ttb, telugu-mtg, turkish-imst, ukrainian-iu, urdu-udtb, uyghur-udt, vietnamese-vtb, wolof-wtb
  - license: CC-BY-SA-NC

- 'jwijffels/udpipe.models.ud.2.3': https://github.com/jwijffels/udpipe.models.ud.2.3
  - UDPipe models constructed on data from Universal Dependencies 2.3
  - languages-treebanks: afrikaans-afribooms, ancient_greek-perseus, ancient_greek-proiel, arabic-padt, armenian-armtdp, basque-bdt, belarusian-hse, bulgarian-btb, catalan-ancora, chinese-gsd, coptic-scriptorium, croatian-set, czech-cac, czech-cltt, czech-fictree, czech-pdt, danish-ddt, dutch-alpino, dutch-lassysmall, english-ewt, english-gum, english-lines, english-partut, estonian-edt, finnish-ftb, finnish-tdt, french-gsd, french-partut, french-sequoia, french-spoken, galician-ctg, galician-treegal, german-gsd, gothic-proiel, greek-gdt, hebrew-htb, hindi-hdtb, hungarian-szeged, indonesian-gsd, irish-idt, italian-isdt, italian-partut, italian-postwita, japanese-gsd, korean-gsd, korean-kaist, latin-ittb, latin-perseus, latin-proiel, latvian-lvtb, lithuanian-hse, maltese-mudt, marathi-ufal, north_sami-giella, norwegian-bokmaal, norwegian-nynorsk, norwegian-nynorsklia, old_church_slavonic-proiel, old_french-srcmf, persian-seraji, polish-lfg, polish-sz, portuguese-bosque, portuguese-gsd, romanian-nonstandard, romanian-rrt, russian-gsd, russian-syntagrus, russian-taiga, serbian-set, slovak-snk, slovenian-ssj, slovenian-sst, spanish-ancora, spanish-gsd, swedish-lines, swedish-talbanken, tamil-ttb, telugu-mtg, turkish-imst, ukrainian-iu, urdu-udtb, uyghur-udt, vietnamese-vtb
  - license: CC-BY-SA-NC

- 'jwijffels/udpipe.models.ud.2.0': https://github.com/jwijffels/udpipe.models.ud.2.0
  - UDPipe models constructed on data from Universal Dependencies 2.0
  - languages-treebanks: ancient_greek-proiel, ancient_greek, arabic, basque, belarusian, bulgarian, catalan, chinese, coptic, croatian, czech-cac, czech-cltt, czech, danish, dutch-lassysmall, dutch, english-lines, english-partut, english, estonian, finnish-ftb, finnish, french-partut, french-sequoia, french, galician-treegal, galician, german, gothic, greek, hebrew, hindi, hungarian, indonesian, irish, italian, japanese, kazakh, korean, latin-ittb, latin-proiel, latin, latvian, lithuanian, norwegian-bokmaal, norwegian-nynorsk, old_church_slavonic, persian, polish, portuguese-br, portuguese, romanian, russian-syntagrus, russian, sanskrit,

slovak, slovenian-sst, slovenian, spanish-ancora, spanish, swedish-lines, swedish, tamil, turkish, ukrainian, urdu, uyghur, vietnamese
  - license: CC-BY-SA-NC
- 'jwijffels/udpipe.models.conll18.baseline': https://github.com/jwijffels/udpipe.models.conll18.baseline
  - UDPipe models constructed on data from Universal Dependencies 2.2
  - languages-treebanks: afrikaans-afribooms, ancient_greek-perseus, ancient_greek-proiel, arabic-padt, armenian-armtdp, basque-bdt, bulgarian-btb, buryat-bdt, catalan-ancora, chinese-gsd, croatian-set, czech-cac, czech-fictree, czech-pdt, danish-ddt, dutch-alpino, dutch-lassysmall, english-ewt, english-gum, english-lines, estonian-edt, finnish-ftb, finnish-tdt, french-gsd, french-sequoia, french-spoken, galician-ctg, galician-treegal, german-gsd, gothic-proiel, greek-gdt, hebrew-htb, hindi-hdtb, hungarian-szeged, indonesian-gsd, irish-idt, italian-isdt, italian-postwita, japanese-gsd, kazakh-ktb, korean-gsd, korean-kaist, kurmanji-mg, latin-ittb, latin-perseus, latin-proiel, latvian-lvtb, mixed, north_sami-giella, norwegian-bokmaal, norwegian-nynorsk, norwegian-nynorsklia, old_church_slavonic-proiel, old_french-srcmf, persian-seraji, polish-lfg, polish-sz, portuguese-bosque, romanian-rrt, russian-syntagrus, russian-taiga, serbian-set, slovak-snk, slovenian-ssj, slovenian-sst, spanish-ancora, swedish-lines, swedish-talbanken, turkish-imst, ukrainian-iu, upper_sorbian-ufal, urdu-udtb, uyghur-udt, vietnamese-vtb
  - license: CC-BY-SA-NC
- 'bnosac/udpipe.models.ud': https://github.com/bnosac/udpipe.models.ud
  - UDPipe models constructed on data from Universal Dependencies 2.1
  - This repository contains models build with this R package on open data from Universal Dependencies 2.1 which allows for commercial usage. The license of these models is mostly CC-BY-SA. Visit that github repository for details on the licenses of the language of your choice. And contact www.bnosac.be if you need support on these models or require models tuned to your needs.
  - languages-treebanks: afrikaans, croatian, czech-cac, dutch, english, finnish, french-sequoia, irish, norwegian-bokmaal, persian, polish, portuguese, romanian, serbian, slovak, spanish-ancora, swedish
  - license: license is treebank-specific but mainly CC-BY-SA and GPL-3 and LGPL-LR
- If you need to train models yourself for commercial purposes or if you want to improve models, you can easily do this with `udpipe_train` which is explained in detail in the package vignette.

Note that when you download these models, you comply to the license of your specific language model.

Value

A data.frame with 1 row and the following columns:

- language: The language as provided by the input parameter `language`
- file_model: The path to the file on disk where the model was downloaded to
- url: The URL where the model was downloaded from
- download_failed: A logical indicating if the download has failed or not due to internet connectivity issues

- download_message: A character string with the error message in case the downloading of the model failed

References

https://ufal.mff.cuni.cz/udpipe, https://github.com/jwijffels/udpipe.models.ud.2.4, https://github.com/jwijffels/udpipe.models.ud.2.3, https://github.com/jwijffels/udpipe.models.conll18.baseline https://github.com/jwijffels/udpipe.models.ud.2.0, https://github.com/bnosac/udpipe.models.ud

See Also

udpipe_load_model

Examples

```
## Not run:
x <- udpipe_download_model(language = "dutch-alpino")
x <- udpipe_download_model(language = "dutch-lassysmall")
x <- udpipe_download_model(language = "russian")
x <- udpipe_download_model(language = "french")
x <- udpipe_download_model(language = "english-partut")
x <- udpipe_download_model(language = "english-ewt")
x <- udpipe_download_model(language = "german-gsd")
x <- udpipe_download_model(language = "spanish-gsd")
x <- udpipe_download_model(language = "spanish-gsd", overwrite = FALSE)

x <- udpipe_download_model(language = "dutch-alpino", udpipe_model_repo = "udpipe.models.ud.2.4")
x <- udpipe_download_model(language = "dutch-alpino", udpipe_model_repo = "udpipe.models.ud.2.3")
x <- udpipe_download_model(language = "dutch-alpino", udpipe_model_repo = "udpipe.models.ud.2.0")
x <- udpipe_download_model(language = "english", udpipe_model_repo = "bnosac/udpipe.models.ud")
x <- udpipe_download_model(language = "dutch", udpipe_model_repo = "bnosac/udpipe.models.ud")
x <- udpipe_download_model(language = "afrikaans", udpipe_model_repo = "bnosac/udpipe.models.ud")
x <- udpipe_download_model(language = "spanish-ancora",
                           udpipe_model_repo = "bnosac/udpipe.models.ud")
x <- udpipe_download_model(language = "dutch-ud-2.1-20180111.udpipe",
                           udpipe_model_repo = "bnosac/udpipe.models.ud")
x <- udpipe_download_model(language = "english",
                          udpipe_model_repo = "jwijffels/udpipe.models.conll18.baseline")

## End(Not run)

x <- udpipe_download_model(language = "sanskrit",
                           udpipe_model_repo = "jwijffels/udpipe.models.ud.2.0",
                           model_dir = tempdir())
x
## cleanup for CRAN
if(file.exists(x$file_model)) file.remove(x$file_model)
```

udpipe_load_model *Load an UDPipe model*

Description

Load an UDPipe model so that it can be use in udpipe_annotate

Usage

```
udpipe_load_model(file)
```

Arguments

file	full path to the model or the value returned by a call to udpipe_download_model

Value

An object of class udpipe_model which is a list with 2 elements

- file: The path to the model as provided by file
- model: An Rcpp-generated pointer to the loaded model which can be used in udpipe_annotate

References

https://ufal.mff.cuni.cz/udpipe

See Also

udpipe_annotate, udpipe_download_model, udpipe_train

Examples

```
## Not run:
x <- udpipe_download_model(language = "dutch-lassysmall")
x$file_model
ud_english <- udpipe_load_model(x$file_model)

x <- udpipe_download_model(language = "english")
x$file_model
ud_english <- udpipe_load_model(x$file_model)

x <- udpipe_download_model(language = "hebrew")
x$file_model
ud_hebrew <- udpipe_load_model(x$file_model)

## End(Not run)


x <- udpipe_download_model(language = "dutch-lassysmall", model_dir = tempdir())
```

```
x$file_model
if(!x$download_failed){
  ud_dutch <- udpipe_load_model(x$file_model)
}

## cleanup for CRAN
if(file.exists(x$file_model)) file.remove(x$file_model)
```

---

udpipe_read_conllu *Read in a CONLL-U file as a data.frame*

Description

Read in a CONLL-U file as a data.frame

Usage

```
udpipe_read_conllu(file)
```

Arguments

file	a connection object or a character string with the location of the file

Value

a data.frame with columns doc_id, paragraph_id, sentence_id, sentence, token_id, token, lemma, upos, xpos, feats, head_token_id, deprel, dep_rel, misc

Examples

```
file_conllu <- system.file(package = "udpipe", "dummydata", "traindata.conllu")
x <- udpipe_read_conllu(file_conllu)
head(x)
```

---

udpipe_train *Train a UDPipe model*

Description

Train a UDPipe model which allows to do Tokenization, Parts of Speech Tagging, Lemmatization and Dependency Parsing or a combination of those.

This function allows you to build models based on data in in CONLL-U format as described at http://universaldependencies.org/format.html. At the time of writing open data in CONLL-U format for 50 languages are available at http://universaldependencies.org/#ud-treebanks.

Most of these are distributed under the CC-BY-SA licence or the CC-BY-NC-SA license.

This function allows to build annotation tagger models based on these data in CONLL-U format, allowing you to have your own tagger model. This is relevant if you want to tune the tagger to your needs or if you don't want to use ready-made models provided under the CC-BY-NC-SA license as shown at udpipe_load_model

Usage

```
udpipe_train(file = file.path(getwd(), "my_annotator.udpipe"),
  files_conllu_training, files_conllu_holdout = character(),
  annotation_tokenizer = "default", annotation_tagger = "default",
  annotation_parser = "default")
```

Arguments

file	full path where the model will be saved. The model will be stored as a binary file which udpipe_load_model can handle. Defaults to 'my_annotator.udpipe' in the current working directory.
files_conllu_training	a character vector of files in CONLL-U format used for training the model
files_conllu_holdout	a character vector of files in CONLL-U format used for holdout evalution of the model. This argument is optional.
annotation_tokenizer	a string containing options for the tokenizer. This can be either 'none' or 'default' or a list of options as mentioned in the UDPipe manual. See the vignette vignette("udpipe-train", package = "udpipe") or go directly to http://ufal.mff.cuni.cz/udpipe/users-manual#model_training_tokenizer for a full description of the options or see the examples below. Defaults to 'default'. If you specify 'none', the model will not be able to perform tokenization.
annotation_tagger	a string containing options for the pos tagger and lemmatiser. This can be either 'none' or 'default' or a list of options as mentioned in the UDPipe manual. See the vignette vignette("udpipe-train", package = "udpipe") or go directly to http://ufal.mff.cuni.cz/udpipe/users-manual#model_training_tagger for a full description of the options or see the examples below. Defaults to 'default'. If you specify 'none', the model will not be able to perform POS tagging or lemmatization.
annotation_parser	a string containing options for the dependency parser. This can be either 'none' or 'default' or a list of options as mentioned in the UDPipe manual. See the vignette vignette("udpipe-train", package = "udpipe") or go directly to http://ufal.mff.cuni.cz/udpipe/users-manual#model_training_parser for a full description of the options or see the examples below. Defaults to 'default'. If you specify 'none', the model will not be able to perform dependency parsing.

Details

In order to train a model, you need to provide files which are in CONLL-U format in argument `files_conllu_training`. This can be a vector of files or just one file. If you do not have your own CONLL-U files, you can download files for your language of choice at http://universaldependencies.org/#ud-treebanks.

At the time of writing open data in CONLL-U format for 50 languages are available at http://universaldependencies.org/#ud-treebanks, namely for: ancient_greek, arabic, basque, belarusian, bulgarian, catalan, chinese, coptic, croatian, czech, danish, dutch, english, estonian, finnish, french, galician, german, gothic, greek, hebrew, hindi, hungarian, indonesian, irish, italian, japanese, kazakh, korean, latin, latvian, lithuanian, norwegian, old_church_slavonic, persian, polish, portuguese, romanian, russian, sanskrit, slovak, slovenian, spanish, swedish, tamil, turkish, ukrainian, urdu, uyghur, vietnamese.

Value

A list with elements

- file: The path to the model, which can be used in `udpipe_load_model`
- annotation_tokenizer: The input argument `annotation_tokenizer`
- annotation_tagger: The input argument `annotation_tagger`
- annotation_parser: The input argument `annotation_parser`
- errors: Messages from the UDPipe process indicating possible errors for example when passing the wrong arguments to the annotation_tokenizer, annotation_tagger or annotation_parser

References

http://ufal.mff.cuni.cz/udpipe/users-manual

See Also

`udpipe_annotation_params`, `udpipe_annotate`, `udpipe_load_model`, `udpipe_accuracy`

Examples

```
## You need to have a file on disk in CONLL-U format, taking the toy example file put in the package
file_conllu <- system.file(package = "udpipe", "dummydata", "traindata.conllu")
file_conllu
cat(head(readLines(file_conllu), 3), sep="\n")

## Not run:
##
## This is a toy example showing how to build a model, it is not a good model whatsoever,
##   because model building takes more than 5 seconds this model is saved also in
##   the file at system.file(package = "udpipe", "dummydata", "toymodel.udpipe")
##
m <- udpipe_train(file = "toymodel.udpipe", files_conllu_training = file_conllu,
  annotation_tokenizer = list(dimension = 16, epochs = 1, batch_size = 100, dropout = 0.7),
   annotation_tagger = list(iterations = 1, models = 1,
```

```
      provide_xpostag = 1, provide_lemma = 0, provide_feats = 0,
      guesser_suffix_rules = 2, guesser_prefix_min_count = 2),
    annotation_parser = list(iterations = 2,
      embedding_upostag = 20, embedding_feats = 20, embedding_xpostag = 0, embedding_form = 50,
      embedding_lemma = 0, embedding_deprel = 20, learning_rate = 0.01,
      learning_rate_final = 0.001, l2 = 0.5, hidden_layer = 200,
      batch_size = 10, transition_system = "projective", transition_oracle = "dynamic",
      structured_interval = 10))

## End(Not run)

file_model <- system.file(package = "udpipe", "dummydata", "toymodel.udpipe")
ud_toymodel <- udpipe_load_model(file_model)
x <- udpipe_annotate(object = ud_toymodel, x = "Ik ging deze morgen naar de bakker brood halen.")
x <- as.data.frame(x)

##
## The above was a toy example showing how to build a model, if you want real-life scenario's
## look at the training parameter examples given below and train it on your CONLL-U file
##
## Example training arguments used for the models available at udpipe_download_model
data(udpipe_annotation_params)
head(udpipe_annotation_params$tokenizer)
head(udpipe_annotation_params$tagger)
head(udpipe_annotation_params$parser)
## Not run:
## More details in the package vignette:
vignette("udpipe-train", package = "udpipe")

## End(Not run)
```

`unique_identifier` *Create a unique identifier for each combination of fields in a data frame*

Description

Create a unique identifier for each combination of fields in a data frame. This unique identifier is unique for each combination of the elements of the fields. The generated identifier is like a primary key or a secondary key on a table. This is just a small wrapper around `frank`

Usage

```
unique_identifier(x, fields, start_from = 1L)
```

Arguments

`x`	a data.frame
`fields`	a character vector of columns from `x`
`start_from`	integer number indicating to start from that number onwards

Value

an integer vector of the same length as the number of rows in x containing the unique identifier

Examples

```
data(brussels_reviews_anno)
x <- brussels_reviews_anno
x$doc_sent_id <- unique_identifier(x, fields = c("doc_id", "sentence_id"))
head(x, 15)
range(x$doc_sent_id)
x$doc_sent_id <- unique_identifier(x, fields = c("doc_id", "sentence_id"), start_from = 10)
head(x, 15)
range(x$doc_sent_id)
```

Index