# Package 'udpipe'

December 7, 2017

**Type** Package

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

**Version** 0.2.2

**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://github.com/bnosac/udpipe

**Encoding** UTF-8

**Depends** R (>= 2.10)

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

**LinkingTo** Rcpp

**Suggests** knitr, topicmodels

**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 [cph],
Jana Straková [cph]

**Repository** CRAN

**Date/Publication** 2017-12-07 13:08:12 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, deprel, dep_rel, misc

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

## See Also

udpipe_annotate

## Examples

```
x <- udpipe_download_model(language = "dutch-lassysmall")
ud_dutch <- udpipe_load_model(x$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
file.remove("dutch-lassysmall-ud-2.0-170801.udpipe")
```

---

# 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)
```

## 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)
```

---

`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

```
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

- ngram: the number of terms which are combined
- collocation: the terms which are combined
- left: the left term of the collocation
- right: the right term of the collocation
- n: the number of times the collocation occurred in the data
- n_left: the number of times the left element of the collocation occurred in the data
- n_right: the number of times the right element of the collocation occurrend 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 <- 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 <- collocation(x, term = "lemma", group = c("doc_id", "sentence_id"),
                      ngram_max = 2, n_min = 2)
head(colloc)
```

`cooccurrence` *Create a cooccurence data.frame*

## Description

A cooccurence data.frame indicates how many times each term co-occurs with another term. This type of dataset is a data.frame with fields term1, term2 and cooc where cooc indicates how many times term1 and term2 co-occurred.
The dataset can be constructed based upon a data frame where you look within a group if 2 terms occurred.
It also can be constructed based upon a vector of words in which case we look how many times each word is followed by another word.

## Usage

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

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

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

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

## Arguments

| | |
|---|---|
| `x` | either<br>• a data.frame where the data.frame contains 1 row per document/term, in which case you need to provide `group` and `term`. This uses cooccurrence.data.frame.<br>• a character vector with terms. This uses cooccurrence.character.<br>• 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 |
| `group` | character string with a column in the data frame `x`. To be used if `x` is a data.frame. |
| `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. |

## 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)

## 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.<br>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 document, term and the summed freq. If freq is not in the dataset, will assume that freq is 1 for each row in 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_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, ...)

## S3 method for class 'data.frame'
document_term_matrix(x, vocabulary, ...)

## 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 document, term and freq indicating how many times a term occurred in that specific document. This is what `document_term_frequencies` returns. |
| `vocabulary` | a character vector of terms which should be present in the document term matrix even if they did not occur in the `x` |

... 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)

## 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)
```

## 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)
```

### Arguments

| | |
|---|---|
| dtm | an object returned by document_term_matrix or an object of class DocumentTermMatrix |
| 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. |

### Value

a sparse Matrix as returned by sparseMatrix or an object of class DocumentTermMatrix 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)
```

---

## 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)
```

### Arguments

| | |
|---|---|
| dtm | an object returned by document_term_matrix or an object of class DocumentTermMatrix |
| terms | a character vector of terms which are in colnames(dtm) and which should be removed |

## Value

a sparse Matrix as returned by `sparseMatrix` or an object of class `DocumentTermMatrix` 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)
dtm <- dtm_remove_terms(dtm, terms = c("appartement", "casa", "centrum", "ciudad"))
dim(dtm)
```

---

`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)
```

## Arguments

| | |
|---|---|
| `dtm` | an object returned by `document_term_matrix` or an object of class DocumentTermMatrix |
| `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 |

## Value

a sparse Matrix as returned by `sparseMatrix` or an object of class `DocumentTermMatrix` 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)

## 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))
```

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

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

## 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.<br>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. |

## 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

a data.frame with columns

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

## 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 <- phrases(x$xpos, pattern = c("DT", "NN", "VB", "RB", "JJ"), sep = "-")
head(np)
np <- phrases(x$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 <- phrases(x$phrase_tag, term = x$token,
                       pattern = "(A|N)+N(P+D*(A|N)*N)*", is_regex = TRUE, ngram_max = 4)
head(nounphrases, 10)
head(sort(table(nounphrases$phrase), 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$pos_sequence)))
np <- phrases(x$xpos, term = x$token, pattern = c("IN", "DT", "NN"))
head(np)
```

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)
```

---

# 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_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*

---

## 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)
```

---

# 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_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_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_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"))
```

udpipe_annotate *Tokenise, Tag and Dependency Parsing Annotation of raw text*

## Description

Tokenise, Tag 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"), ...)
```

## 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. |
| ... | 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

```
x <- udpipe_download_model(language = "dutch-lassysmall")
ud_dutch <- udpipe_load_model(x$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")
as.data.frame(x)


## cleanup for CRAN only - you probably want to keep your model if you have downloaded it
file.remove("dutch-lassysmall-ud-2.0-170801.udpipe")
```

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 50 languages trained on 67 treebanks are provided by UDPipe at https://lindat.mff.cuni.cz/repository/xmlui/handle/11234/1-2364 in one zip file. You can either download these manually in order to use it for annotation purposes or use udpipe_download_model to download these models for a specific language of choice.

For your convenience, these models are also made available at https://github.com/jwijffels/udpipe.models.ud.2.0 under the CC-BY-NC-SA licence. This function downloads the models

from that location, so if you use this function you are complying to that license. If you want to train models for commercial purposes, you can easily do this with udpipe_train

## Usage

```
udpipe_download_model(language = c("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"), model_dir = getwd())
```

## Arguments

| | |
|---|---|
| language | a character stirng with a language. Possible values are: 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 |
| model_dir | a path where the model will be downloaded to. Defaults to the current working directory |

## Details

Pre-trained Universal Dependencies 2.0 models on all UD treebanks are made available at https://ufal.mff.cuni.cz/udpipe, namely at https://lindat.mff.cuni.cz/repository/xmlui/handle/11234/1-2364. At the time of writing this consists of models made available on 50 languages, namely: 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. Mark that these models are made available under the CC BY-NC-SA 4.0 license.

These models are also provided in an R package for your convenience at https://github.com/jwijffels/udpipe.models.ud.2.0

## Value

A data.frame with 1 row and 2 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

## References

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

## See Also

udpipe_load_model

## Examples

```
x <- udpipe_download_model(language = "sanskrit", model_dir = tempdir())
x
x$file_model
## Not run:
x <- udpipe_download_model(language = "dutch")
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")
x <- udpipe_download_model(language = "german")
x <- udpipe_download_model(language = "spanish")

## End(Not run)
```

---

# 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 |

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

## See Also

udpipe_annotate, udpipe_download_model, udpipe_train

## Examples

```
x <- udpipe_download_model(language = "dutch-lassysmall", model_dir = tempdir())
x$file_model
ud_dutch <- udpipe_load_model(x$file_model)
## Not run:
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)
```

---

# 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

## 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