# Package 'cleanNLP'

November 18, 2018

**Type** Package

**Title** A Tidy Data Model for Natural Language Processing

**Version** 2.3.0

**Author** Taylor B. Arnold [aut, cre]

**Maintainer** Taylor B. Arnold <taylor.arnold@acm.org>

**Description** Provides a set of fast tools for converting a textual corpus into a set of normalized tables. Users may make use of the 'udpipe' back end with no external dependencies, a Python back end with 'spaCy' <https://spacy.io> or the Java back end 'CoreNLP' <http://stanfordnlp.github.io/CoreNLP/>. Exposed annotation tasks include tokenization, part of speech tagging, named entity recognition, entity linking, sentiment analysis, dependency parsing, coreference resolution, and word embeddings. Summary statistics regarding token unigram, part of speech tag, and dependency type frequencies are also included to assist with analyses.

**Depends** R (>= 2.10)

**Imports** dplyr (>= 0.7.4), Matrix (>= 1.2), stringi, stats, methods, utils

**Suggests** udpipe (>= 0.3), reticulate (>= 1.4), rJava (>= 0.9-8), RCurl (>= 1.95), knitr (>= 1.15), rmarkdown (>= 1.4), testthat (>= 1.0.1), covr (>= 2.2.2)

**SystemRequirements** Python (>= 2.7.0); spaCy <https://spacy.io/> (>= 2.0); Java (>= 7.0); Stanford CoreNLP <http://nlp.stanford.edu/software/corenlp.shtml> (>= 3.9.2)

**License** LGPL-2

**URL** https://statsmaths.github.io/cleanNLP/

**BugReports** http://github.com/statsmaths/cleanNLP/issues

**LazyData** true

**VignetteBuilder** knitr

**RoxygenNote** 6.1.1

**NeedsCompilation** no

**Repository** CRAN

**Date/Publication** 2018-11-18 15:50:03 UTC

# R topics documented:



---

`cleanNLP-package` *cleanNLP: A Tidy Data Model for Natural Language Processing*

---

## Description

Provides a set of fast tools for converting a textual corpus into a set of normalized tables. The underlying natural language processing pipeline utilizes either the Python module spaCy or the Java-based Stanford CoreNLP library. The Python option is faster and generally easier to install; the Java option has additional annotators that are not available in spaCy.

## Details

Once the package is set up, run one of cnlp_init_tokenizers, cnlp_init_spacy, or cnlp_init_corenlp to load the desired NLP backend. After this function is done running, use cnlp_annotate to run the annotation engine over a corpus of text. Functions are then available to extract data tables from the annotation object: cnlp_get_token, cnlp_get_dependency, cnlp_get_document, cnlp_get_coreference, cnlp_get_entity, cnlp_get_sentence, and cnlp_get_vector. See their documentation for further details. The package vignettes provide more detailed set-up information.

If loading annotation that have previously been saved to disk, these can be pulled back into R using cnlp_read_csv. This does not require Java or Python nor does it require initializing the annotation pipeline.

## Author(s)

**Maintainer**: Taylor B. Arnold <taylor.arnold@acm.org>

## See Also

Useful links:

- https://statsmaths.github.io/cleanNLP/
- Report bugs at http://github.com/statsmaths/cleanNLP/issues

## Examples

```
## Not run:
# load the annotation engine (can also use spaCy and coreNLP backends)
setup_tokenizers_backend()
init_backend(type = "tokenizers")

# annotate your text
annotation <- run_annotators("path/to/corpus/directory")

# pull off data tables
token <- cnlp_get_token(annotation)
dependency <- cnlp_get_dependency(annotation)
document <- cnlp_get_document(annotation)
coreference <- cnlp_get_coreference(annotation)
entity <- cnlp_get_entity(annotation)
sentiment <- cnlp_get_sentence(annotation)
vector <- cnlp_get_vector(annotation)

## End(Not run)
```

cnlp_annotate *Run the annotation pipeline on a set of documents*

## Description

Runs the clean_nlp annotators over a given corpus of text using either the R, Java, or Python backend. The details for which annotators to run and how to run them are specified by using one of: cnlp_init_tokenizers, cnlp_init_spacy, cnlp_init_udpipe, or cnlp_init_corenlp.

## Usage

```
cnlp_annotate(input, as_strings = NULL, doc_ids = NULL,
  backend = NULL, meta = NULL, doc_var = "doc_id",
  text_var = "text")
```

## Arguments

| | |
|---|---|
| input | either a vector of file names to parse, a character vector with one document in each element, or a data frame. If a data frame, specify what column names contain the text and (optionally) document ids |
| as_strings | logical. Is the data given to input the actual document text or are they file names? If NULL, the default, will be set to FALSE if the input points to a valid file and TRUE otherwise. |
| doc_ids | optional character vector of document names |
| backend | which backend to use. Will default to the last model to be initalized. |
| meta | an optional data frame to bind to the document table |
| doc_var | if passing a data frame, character description of the column containing the document identifier; if this this variable does not exist in the dataset, automatic names will be given (or set to NULL to force automatic names) |
| text_var | if passing a data frame, which column contains the document identifier |

## Value

an object of class annotation

## Author(s)

Taylor B. Arnold, <taylor.arnold@acm.org>

## References

Manning, Christopher D., Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. 2014. The Stanford corenlp Natural Language Processing Toolkit. In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations, pp. 55-60.*

## Examples

```
## Not run:
annotation <- cnlp_annotate("path/to/corpus/directory")

## End(Not run)
```

---

`cnlp_combine_documents` *Combine a set of annotations*

---

## Description

Takes an arbitrary set of annotations and efficiently combines them into a single object. All document ids are reset so that they are contiguous integers starting at zero.

## Usage

```
cnlp_combine_documents(...)
```

## Arguments

| | |
|---|---|
| `...` | annotation objects to combine; either a single list item or all of the objects as individual inputs |

## Value

a single annotation object containing all of the input documents

## Author(s)

Taylor B. Arnold, <taylor.arnold@acm.org>

## Examples

```
## Not run:
annotation <- combine_annotators(anno01, anno02, anno03)

## End(Not run)
```

`cnlp_download_corenlp` *Download java files needed for CoreNLP*

## Description

The cleanNLP package does not supply the raw java files provided by the Stanford NLP Group as they are quite large. This function downloads the libraries automatically, by default into the directory where the package was installed. These are not required for using the spaCy Python implementation.

## Usage

```
cnlp_download_corenlp(type = c("default", "base", "en", "fr", "de", "es",
  "ar", "zh"), output_loc, url = NULL, url_core = TRUE,
  force = FALSE)
```

## Arguments

| | |
|---|---|
| `type` | type of files to download. The base package Other jars include model files for French, German, and Spanish. These can be installed in addition to the base package. By default, the function downloads the base package and English model files. |
| `output_loc` | a string showing where the files are to be downloaded. If missing, will try to download files into the directory where the package was original installed. |
| `url` | the url to try to download components from. Setting to NULL uses the default location on the Stanford NLP server, but you can set this manually by using this option. It also allows for local files, but note that the path must include the prefix `file://`. For details, see `download.file`. |
| `url_core` | if `url` is not null, this flag indicates whether the path given to url points to the core nlp files (which are zipped) or to model files (which are unzipped). |
| `force` | logical. Should files be downloaded if they appear to already exist? |

## Examples

```
## Not run:
cnlp_download_corenlp()
cnlp_download_corenlp(type="spanish")

## End(Not run)
```

## cnlp_download_udpipe *Download model files needed for udpipe*

### Description

The cleanNLP package does not supply the model files required for using the udpipe backend. These files can be downloaded with this function. The models are saved, by default, in the location where the package is installed. They will be saved between R sessions. This function is called internally by `cnlp_init_udpipe` if a model is not available.

### Usage

```
cnlp_download_udpipe(model_name = "english", model_loc = NULL)
```

### Arguments

| | |
|---|---|
| model_name | string giving the model namel. Defaults to "english" if NULL. |
| model_loc | where should be model be downloaded to. If set to NULL, will be downloaded in the location that the cleanNLP package is installed. |

### Examples

```
## Not run:
cnlp_download_core_nlp()
cnlp_download_core_nlp(type="spanish")

## End(Not run)
```

## cnlp_extract_documents *Extract documents from an annotation object*

### Description

Takes an annotation object and returns an annotation object containing only a subset of the documents.

### Usage

```
cnlp_extract_documents(anno, ids)
```

### Arguments

| | |
|---|---|
| anno | the object to extract from |
| ids | a vector of document ids from which to extract |

### Value

a new annotation object

### Author(s)

Taylor B. Arnold, <taylor.arnold@acm.org>

### Examples

```
## Not run:
annotation <- extract_documents(anno, ids = c(0, 1, 2, 3))

## End(Not run)
```

---

## cnlp_get_coreference *Access coreferences from an annotation object*

---

### Description

Coreferences are collections of expressions that all represent the same person, entity, or thing. For example, the text "Lauren loves dogs. She would walk them all day.", there is a coreference consisting of the token "Lauren" in the first sentence and the token "She" in the second sentence. In the output given from this function, a row is given for any mention of an entity; these can be linked using the `rid` key.

### Usage

```
cnlp_get_coreference(annotation)
```

### Arguments

| | |
|---|---|
| `annotation` | an annotation object |

### Value

Returns an object of class `c("tbl_df", "tbl", "data.frame")` containing one row for every coreference in the corpus.

The returned data frame includes at least the following columns:

- "id" - integer. Id of the source document.
- "rid" - integer. Relation ID.
- "mid" - integer. Mention ID; unique to each coreference within a document.
- "mention" - character. The mention as raw words from the text.
- "mention_type" - character. One of "LIST", "NOMINAL", "PRONOMINAL", or "PROPER".
- "number" - character. One of "PLURAL", "SINGULAR", or "UNKNOWN".

- "gender" - character. One of "FEMALE", "MALE", "NEUTRAL", or "UNKNOWN".
- "animacy" - character. One of "ANIMATE", "INANIMATE", or "UNKNOWN".
- "sid" - integer. Sentence id of the coreference.
- "tid" - integer. Token id at the start of the coreference.
- "tid_end" - integer. Token id at the start of the coreference.
- "tid_head" - integer. Token id of the head of the coreference.

## Author(s)

Taylor B. Arnold, <taylor.arnold@acm.org>

## References

Manning, Christopher D., Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. 2014. The Stanford CoreNLP Natural Language Processing Toolkit. In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations, pp. 55-60.*

Marta Recasens, Marie-Catherine de Marneffe, and Christopher Potts. The Life and Death of Discourse Entities: Identifying Singleton Mentions. In: *Proceedings of NAACL 2013*.

Heeyoung Lee, Angel Chang, Yves Peirsman, Nathanael Chambers, Mihai Surdeanu and Dan Jurafsky. Deterministic coreference resolution based on entity-centric, precision-ranked rules. Computational Linguistics 39(4), 2013.

Heeyoung Lee, Yves Peirsman, Angel Chang, Nathanael Chambers, Mihai Surdeanu, Dan Jurafsky. Stanford's Multi-Pass Sieve Coreference Resolution System at the CoNLL-2011 Shared Task. In: *Proceedings of the CoNLL-2011 Shared Task, 2011*.

Karthik Raghunathan, Heeyoung Lee, Sudarshan Rangarajan, Nathanael Chambers, Mihai Surdeanu, Dan Jurafsky, Christopher Manning A Multi-Pass Sieve for Coreference Resolution. EMNLP-2010, Boston, USA. 2010.

---

# cnlp_get_dependency *Access dependencies from an annotation object*

---

## Description

This function grabs the table of dependencies from an annotation object. These are binary relationships between the tokens of a sentence. Common examples include nominal subject (linking the object of a sentence to a verb), and adjectival modifiers (linking an adjective to a noun). While not included in the underlying data, the function has an option for linking these dependencies to the raw words and lemmas in the table of tokens. Both language-agnostic and language-specific universal dependency types are included in the output.

## Usage

```
cnlp_get_dependency(annotation, get_token = FALSE)
```

## Arguments

| | |
|---|---|
| annotation | an annotation object |
| get_token | logical. Should words and lemmas be attached to the returned dependency table. |

## Value

Returns an object of class c("tbl_df", "tbl", "data.frame") containing one row for every dependency pair in the corpus.

The returned data frame includes at a minimum the following columns:

- "id" - integer. Id of the source document.
- "sid" - integer. Sentence id of the source token.
- "tid" - integer. Id of the source token.
- "tid_target" - integer. Id of the source token.
- "relation" - character. Language-agnostic universal dependency type.
- "relation_full" - character. Language specific universal dependency type.

If cnlp_get_token is set to true, the following columns will also be included:

- "word" - character. The source word in the raw text.
- "lemma" - character. Lemmatized form of the source word.
- "word_target" - character. The target word in the raw text.
- "lemma_target" - character. Lemmatized form of the target word.

## Author(s)

Taylor B. Arnold, <taylor.arnold@acm.org>

## References

Manning, Christopher D., Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. 2014. The Stanford CoreNLP Natural Language Processing Toolkit. In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations, pp. 55-60.*

Danqi Chen and Christopher D Manning. 2014. A Fast and Accurate Dependency Parser using Neural Networks. In: *Proceedings of EMNLP 2014*

Spence Green, Marie-Catherine de Marneffe, John Bauer, and Christopher D. Manning. 2010. Multiword Expression Identification with Tree Substitution Grammars: A Parsing tour de force with French. In: *EMNLP 2011*.

Spence Green and Christopher D. Manning. 2010. Better Arabic Parsing: Baselines, Evaluations, and Analysis. In: COLING 2010.

Pi-Chuan Chang, Huihsin Tseng, Dan Jurafsky, and Christopher D. Manning. 2009. Discriminative Reordering with Chinese Grammatical Relations Features. In: Proceedings of the Third Workshop on Syntax and Structure in Statistical Translation.

Anna Rafferty and Christopher D. Manning. 2008. Parsing Three German Treebanks: Lexicalized and Unlexicalized Baselines. In: *ACL Workshop on Parsing German.*

## Examples

```
data(obama)

# find the most common noun lemmas that are the syntactic subject of a
# clause
require(dplyr)
res <- cnlp_get_dependency(obama, get_token = TRUE) %>%
  filter(relation == "nsubj")
res$lemma_target %>%
  table() %>%
  sort(decreasing = TRUE) %>%
  head(n = 40)
```

---

# cnlp_get_document *Access document meta data from an annotation object*

---

## Description

Access document meta data from an annotation object

## Usage

```
cnlp_get_document(annotation)
```

## Arguments

| | |
|---|---|
| annotation | an annotation object |

## Value

Returns an object of class `c("tbl_df", "tbl", "data.frame")` containing one row for every document in the corpus.

The returned data frame includes at least the following columns:

- "id" - integer. Id of the source document.
- "time" - date time. The time at which the parser was run on the text.
- "version" - character. Version of the CoreNLP library used to parse the text.
- "language" - character. Language of the text, in ISO 639-1 format.
- "uri" - character. Description of the raw text location. Set to `NA` if parsed from in-memory character vector.

Other application specific columns may be included as additional variables.

## Author(s)

Taylor B. Arnold, <taylor.arnold@acm.org>

## References

Manning, Christopher D., Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. 2014. The Stanford CoreNLP Natural Language Processing Toolkit. In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations, pp. 55-60.*

## Examples

```
data(obama)

cnlp_get_document(obama)
```

---

`cnlp_get_entity` *Access named entities from an annotation object*

---

## Description

Named entity recognition attempts to find the mentions of various categories within the corpus of text. Common example include proper references to location (e.g., "Boston", or "England") or people (e.g., "Winston Churchill"), as well as specific dates (e.g., "tomorrow", or "September 19th") times, or numbers.

## Usage

```
cnlp_get_entity(annotation)
```

## Arguments

| | |
|---|---|
| `annotation` | an annotation object |

## Details

When using CoreNLP, the default entity types are:

- "LOCATION" Countries, cities, states, locations, mountain ranges, bodies of water.
- "PERSON" People, including fictional.
- "ORGANIZATION" Companies, agencies, institutions, etc.
- "MONEY" Monetary values, including unit.
- "PERCENT" Percentages.
- "DATE" Absolute or relative dates or periods.
- "TIME" Times smaller than a day.

For the spaCy engine there is no generic LOCATION, ORGANIZATION is shortened to ORG, and the following categories are added:

- "NORP" Nationalities or religious or political groups.
- "FACILITY" Buildings, airports, highways, bridges, etc.
- "GPE" Countries, cities, states.
- "LOC" Non-GPE locations, mountain ranges, bodies of water.
- "PRODUCT" Objects, vehicles, foods, etc. (Not services.)
- "EVENT" Named hurricanes, battles, wars, sports events, etc.
- "WORK_OF_ART" Titles of books, songs, etc.
- "LANGUAGE" Any named language.
- "QUANTITY" Measurements, as of weight or distance.
- "ORDINAL" "first", "second", etc.
- "CARDINAL" Numerals that do not fall under another type.

## Value

Returns an object of class `c("tbl_df", "tbl", "data.frame")` containing one row for every named entity mention in the corpus.

The returned data frame includes the following columns:

- "id" - integer. Id of the source document.
- "sid" - integer. Sentence id of the entity mention.
- "tid" - integer. Token id at the start of the entity mention.
- "tid_end" - integer. Token id at the end of the entity mention.
- "entity_type" - character. See below from details.
- "entity" - character. Raw words of the named entity in the text.

## Author(s)

Taylor B. Arnold, `<taylor.arnold@acm.org>`

## References

Manning, Christopher D., Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. 2014. The Stanford CoreNLP Natural Language Processing Toolkit. In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations, pp. 55-60.*

Jenny Rose Finkel, Trond Grenager, and Christopher Manning. 2005. Incorporating Non-local Information into Information Extraction Systems by Gibbs Sampling. In: *Proceedings of the 43nd Annual Meeting of the Association for Computational Linguistics (ACL 2005), pp. 363-370.*

## Examples

```
require(dplyr)
data(obama)

# what are the most common entity types used in the addresses?
cnlp_get_entity(obama)$entity_type %>%
 table()

# what are the most common locations mentioned?
res <- cnlp_get_entity(obama) %>%
  filter(entity_type == "LOCATION")
res$entity %>%
  table() %>%
  sort(decreasing = TRUE) %>%
  head(n = 25)

# what are the most common organizations mentioned?
res <- cnlp_get_entity(obama) %>%
  filter(entity_type == "ORGANIZATION")
res$entity %>%
  table() %>%
  sort(decreasing = TRUE) %>%
  head(n = 25)
```

---

# cnlp_get_sentence *Access sentence-level annotations*

---

## Description

Access sentence-level annotations

## Usage

```
cnlp_get_sentence(annotation)
```

## Arguments

| | |
|---|---|
| annotation | an annotation object |

## Value

Returns an object of class `c("tbl_df", "tbl", "data.frame")` containing one row for every sentence in the corpus.

The returned data frame includes at a minimum the following columns:

- "id" - integer. Id of the source document.
- "sid" - integer. Sentence id.

The coreNLP backend also currently returns a column "sentiment" that gives a score from 0 (most negative) to 4 (most positive) for how positive the tone of the sentence is predicted to be.

## Author(s)

Taylor B. Arnold, <taylor.arnold@acm.org>

## References

Manning, Christopher D., Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. 2014. The Stanford CoreNLP Natural Language Processing Toolkit. In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations, pp. 55-60.*

Socher, Richard, et al. "Recursive deep models for semantic compositionality over a sentiment treebank." Proceedings of the conference on empirical methods in natural language processing (EMNLP). Vol. 1631. 2013.

---

# cnlp_get_token *Access tokens from an annotation object*

---

## Description

This function grabs the table of tokens from an annotation object. There is exactly one row for each token found in the raw text. Tokens include words as well as punctuation marks. If `include_root` is set to `TRUE`, a token called `ROOT` is also added to each sentence; it is particularly useful when interacting with the table of dependencies.

## Usage

```
cnlp_get_token(annotation, include_root = FALSE, combine = FALSE,
  remove_na = combine, spaces = FALSE)
```

## Arguments

| | |
|---|---|
| `annotation` | an annotation object |
| `include_root` | boolean. Should the sentence root be included? Set to FALSE by default. |
| `combine` | boolean. Should other tables (dependencies, sentences, and entites) by merge with the tokens? Set to FALSE by default. |
| `remove_na` | boolean. Should columns with only non-missing values be removed? This is mostly useful when working with the combine options, and by default is equal to whatever `combine` is set to. |
| `spaces` | should a column be included that gives the number of spaces that should come after the word. Useful for reconstructing the original text. |

## Value

Returns an object of class `c("tbl_df", "tbl", "data.frame")` containing one row for every token in the corpus. The root of each sentence is included as its own token.

The returned data frame includes at a minimum the following columns, unless `remove_na` has been selected in which case only the first four columns are guaranteed to be in the output depending on which annotators were run:

- "id" - integer. Id of the source document.
- "sid" - integer. Sentence id, starting from 0.
- "tid" - integer. Token id, with the root of the sentence starting at 0.
- "word" - character. Raw word in the input text.
- "lemma" - character. Lemmatized form the token.
- "upos" - character. Universal part of speech code.
- "pos" - character. Language-specific part of speech code; uses the Penn Treebank codes.
- "cid" - integer. Character offset at the start of the word in the original document.

## Author(s)

Taylor B. Arnold, `<taylor.arnold@acm.org>`

## References

Manning, Christopher D., Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. 2014. The Stanford CoreNLP Natural Language Processing Toolkit. In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations, pp. 55-60.*

Kristina Toutanova and Christopher D. Manning. 2000. Enriching the Knowledge Sources Used in a Maximum Entropy Part-of-Speech Tagger. In: *Proceedings of the Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora (EMNLP/VLC-2000), pp. 63-70.*

Kristina Toutanova, Dan Klein, Christopher Manning, and Yoram Singer. 2003. Feature-Rich Part-of-Speech Tagging with a Cyclic Dependency Network. In: *Proceedings of HLT-NAACL 2003, pp. 252-259.*

## Examples

```
data(obama)

# find average sentence length from each address
require(dplyr)
cnlp_get_token(obama) %>%
  group_by(id, sid) %>%
  summarize(sentence_length = max(tid)) %>%
  summarize(avg_sentence_length = mean(sentence_length))
```

## cnlp_get_vector *Access word embedding vector from an annotation object*

### Description

Word embeddings map each lemma or token into a high-dimensional vector space. The implementation here uses a 300-dimensional space. Only available with the spaCy parser.

### Usage

```
cnlp_get_vector(annotation)
```

### Arguments

| | |
|---|---|
| annotation | an annotation object |

### Value

Returns a matrix containing one row for every triple found in the corpus, or NULL if not embeddings are present

### Author(s)

Taylor B. Arnold, <taylor.arnold@acm.org>

### References

Pennington, Jeffrey, Richard Socher, and Christopher D. Manning. "Glove: Global Vectors for Word Representation." EMNLP. Vol. 14. 2014.

## cnlp_init_corenlp *Interface for initializing the corenlp backend*

### Description

This function must be run before annotating text with the corenlp backend. It sets the properties for the corenlp engine and loads the file using rJava interface provided by reticulate. See Details for more information about the anno_level codes.

### Usage

```
cnlp_init_corenlp(language, anno_level = 2, lib_location = NULL,
  mem = "6g", verbose = FALSE)
```

## Arguments

| | |
|---|---|
| language | a character vector describing the desired language; should be one of: "ar", "de", "en", "es", "fr", or "zh". |
| anno_level | integer code. Sets which annotators should be loaded, based on on how long they take to load and run. anno_level 0 is the fastest, and anno_level 8 is the slowest. See Details for a full description of the levels |
| lib_location | a string giving the location of the corenlp java files. This should point to a directory which contains, for example the file "stanford-corenlp-*.jar", where "*" is the version number. If missing, the function will try to find the library in the environment variable corenlp_HOME, and otherwise will fail. (Java model only) |
| mem | a string giving the amount of memory to be assigned to the rJava engine. For example, "6g" assigned 6 gigabytes of memory. At least 2 gigabytes are recommended at a minimum for running the corenlp package. On a 32bit machine, where this is not possible, setting "1800m" may also work. This option will only have an effect the first time `init_backend` is called for the corenlp backend, and also will not have an effect if the java engine is already started by another process. |
| verbose | boolean. Should messages from the pipeline be written to the console or suppressed? |

## Details

Currently available anno_level codes are integers from 0 to 8. Setting anno_level above 2 has no additional effect on the German and Spanish models. Setting above 1 has no effect on the French model. The available anno_level codes are:

- "0" runs just the tokenizer, sentence splitter, and part of speech tagger. Extremely fast.
- "1" includes the dependency parsers and, for English, the sentiment tagger. Often 20-30x slower than anno_level 0.
- "2" adds the named entity annotator to the parser and sentiment tagger (when available). For English models, it also includes the mentions and natlog annotators. Usually no more than twice as slow as anno_level 1.
- "3" add the coreference resolution annotator to the anno_level 2 annotators. Depending on the corpus, this takes about 2-4x longer than the anno_level 2 annotators

We suggest starting at anno_level 2 and down grading to 0 if your corpus is particularly large, or upgrading to 3 if you sacrifice the slowdown. If your text is not formal written text (i.e., tweets or text messages), the anno_level 0 annotator should still work well but anything beyond that may be difficult. Semi-formal text such as e-mails or transcribed speech are generally okay to run for all of the levels.

## Author(s)

Taylor B. Arnold, <taylor.arnold@acm.org>

## Examples

```
## Not run:
cnlp_init_corenlp("en")

## End(Not run)
```

---

cnlp_init_spacy *Interface for initializing the spacy backend*

---

## Description

This function must be run before annotating text with the spacy backend. It sets the properties for the spacy engine and loads the file using the R to Python interface provided by reticulate.

## Usage

```
cnlp_init_spacy(model_name = NULL, entity_flag = TRUE,
  vector_flag = FALSE)
```

## Arguments

| | |
|---|---|
| model_name | string giving the model name for the Python/spacy backend. Defaults to "en_core_web_sm" (English) if set to NULL. |
| entity_flag | boolean. Should named entities be identified. |
| vector_flag | boolean. Should word vectors be computed and saved. |

## Author(s)

Taylor B. Arnold, <taylor.arnold@acm.org>

## Examples

```
## Not run:
cnlp_init_spacy(vector_flag = TRUE)

## End(Not run)
```

## cnlp_init_tokenizers *Interface for initializing the tokenizers backend*

### Description

This function must be run before annotating text with the tokenizers backend.

### Usage

```
cnlp_init_tokenizers(locale = NULL)
```

### Arguments

| | |
|---|---|
| locale | string giving the locale name to pass to the stringi functions. If NULL, the default locale is selected |

### Author(s)

Taylor B. Arnold, <taylor.arnold@acm.org>

### Examples

```
## Not run:
cnlp_init_tokenizers()

## End(Not run)
```

## cnlp_init_udpipe *Interface for initializing the udpipe backend*

### Description

This function must be run before annotating text with the udpipe backend. It will parse in English by default, but you can load other models as well.

### Usage

```
cnlp_init_udpipe(model_name = NULL, model_path = NULL,
  feature_flag = FALSE, parser = "default")
```

### Arguments

| | |
|---|---|
| model_name | string giving the model namel. Defaults to "english" if NULL. Ignored if model_path is not NULL. |
| model_path | provide a full path to a model file. |
| feature_flag | boolean. Should universal features be included in the output. |
| 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 |

### Author(s)

Taylor B. Arnold, <taylor.arnold@acm.org>

### Examples

```
## Not run:
cnlp_init_udpipe(model_name = "english")

## End(Not run)
```

---

## cnlp_quick *Quickly Compute Data Frame of Annotations*

---

### Description

Runs the clean_nlp annotators over a given corpus of text and returns a data frame of annotated text with one token per line. By default it will initalize the udpipe backend if no annotators are found.

### Usage

```
cnlp_quick(input, ...)
```

### Arguments

| | |
|---|---|
| input | either a vector of file names to parse, a character vector with one document in each element, or a data frame. If a data frame, specify what column names contain the text and (optionally) document ids |
| ... | additional options passed to cnlp_annotate |

### Value

a data frame of annotations with one row per token

### Author(s)

Taylor B. Arnold, <taylor.arnold@acm.org>

## Examples

```
## Not run:
annotation <- cnlp_quick(c("Parse this text.", "This too, as a new doc."))

## End(Not run)
```

---

`cnlp_read_conll` *Reads a CoNLL-U or CoNLL-X File*

---

## Description

Takes an file saved in the CoNLL-U or CoNLL-X format and converts it into an annotation object. This is a lossy procedure, grabbing just tokenization, lemmatization, part of speech tags, and dependencies.

## Usage

```
cnlp_read_conll(file)
```

## Arguments

| | |
|---|---|
| `file` | character vector giving the path to the file |

## Value

an annotation object with a single document

## Author(s)

Taylor B. Arnold, <taylor.arnold@acm.org>

## Examples

```
## Not run:
cnlp_read_conll(annotation, "/path/to/file.conll")

## End(Not run)
```

cnlp_read_csv *Read annotation files from disk*

## Description

Loads an annotation that has been stored as a set of csv files in a local directory. This is typically created by a call to cnlp_annotate or cnlp_write_csv.

## Usage

```
cnlp_read_csv(input_dir)
```

## Arguments

| | |
|---|---|
| input_dir | path to the directory where the files are stored |

## Value

an object of class annotation

## Author(s)

Taylor B. Arnold, <taylor.arnold@acm.org>

## Examples

```
## Not run:
annotation <- cnlp_read_csv("path/to/annotation")

## End(Not run)
```

cnlp_utils_pca *Compute Principal Components and store as a Data Frame*

## Description

Takes a matrix, perhaps from the output of cnlp_utils_tfidf, and returns a data frame with the top principal components extracted. This is a simple but powerful technique for visualizing a corpus of documents.

## Usage

```
cnlp_utils_pca(x, meta = NULL, k = 2, center = TRUE, scale = TRUE)
```

### Arguments

| | |
|---|---|
| x | a matrix object to pass to prcomp |
| meta | an optional object to append to the front of the principal components. Can be a vector or a data frame, but must have the same length or number of rows as the number of rows in x |
| k | integer. The number of components to include in the output. |
| center | logical. Should the data be centered? |
| scale | logical. Should the data be scaled? Note that this will need to be set to false if any columns in x are constant if center is also true. |

### Value

a data_frame object containing the top k principal components of the data in x, with the object meta appended to the front, when it is non-null.

### Examples

```
require(dplyr)
data(obama)

# Get principal components from the non-proper noun lemmas
tfidf <- cnlp_get_token(obama) %>%
  filter(pos %in% c("NN", "NNS")) %>%
  cnlp_utils_tfidf()
pca_doc <- cnlp_utils_pca(tfidf, cnlp_get_document(obama))

# Plot speeches using the first two principal components
plot(pca_doc$PC1, pca_doc$PC2, col = "white")
text(pca_doc$PC1, pca_doc$PC2, label = 2009:2016)
```

---

## cnlp_utils_tfidf *Construct the TF-IDF Matrix from Annotation or Data Frame*

---

### Description

Given an annotation object, this function returns the term-frequency inverse document frequency (tf-idf) matrix from the extracted lemmas. A data frame with a document id column and token column can be also be given, which allows the user to preprocess and filter the desired tokens to include.

### Usage

```
cnlp_utils_tfidf(object, type = c("tfidf", "tf", "idf", "vocab", "all"),
  tf_weight = c("lognorm", "binary", "raw", "dnorm"),
  idf_weight = c("idf", "smooth", "prob"), min_df = 0.1,
  max_df = 0.9, max_features = 10000, doc_var = c("doc_id", "id"),
```

```
  token_var = "lemma", vocabulary = NULL, doc_set = NULL)

cnlp_utils_tf(object, type = "tf", tf_weight = "raw", ...)
```

## Arguments

| | |
|---|---|
| object | either an annotation object or a data frame with columns equal to the inputs given to doc_var and token_var |
| type | the desired return type. The options tfidf, tf, and idf return a list with the desired matrix, the document ids, and the vocabulary set. The option all returns a list with all three as well as the ids and vocabulary. For consistency, vocab all returns a list but this only contains the ids and vocabulary set. |
| tf_weight | the weighting scheme for the term frequency matrix. The selection lognorm takes one plus the log of the raw frequency (or zero if zero), binary encodes a zero one matrix indicating simply whether the token exists at all in the document, raw returns raw counts, and dnorm uses double normalization. |
| idf_weight | the weighting scheme for the inverse document matrix. The selection idf gives the logarithm of the simple inverse frequency, smooth gives the logarithm of one plus the simple inverse frequency, and prob gives the log odds of the the token occurring in a randomly selected document. |
| min_df | the minimum proportion of documents a token should be in to be included in the vocabulary |
| max_df | the maximum proportion of documents a token should be in to be included in the vocabulary |
| max_features | the maximum number of tokens in the vocabulary |
| doc_var | character vector. The name of the column in object that contains the document ids, unless object is an annotation object, in which case it's the column of the token matrix to use as the document id. |
| token_var | character vector. The name of the column in object that contains the tokens, unless object is an annotation object, in which case it's the column of the token matrix to use as the tokens (generally either lemma or word). |
| vocabulary | character vector. The vocabulary set to use in constructing the matrices. Will be computed within the function if set to NULL. When supplied, the options min_df, max_df, and max_features are ignored. |
| doc_set | optional character vector of document ids. Useful to create empty rows in the output matrix for documents without data in the input. Most users will want to keep this equal to NULL, the default, to have the function compute the document set automatically. |
| ... | other arguments passed to the base method |

## Value

a sparse matrix with dimnames or, if "all", a list with elements

- tf the term frequency matrix
- idf the inverse document frequency matrix

- tfidf the product of the tf and idf matrices
- vocab a character vector giving the vocabulary used in the function, corresponding to the columns of the matrices
- id a vector of the doc ids, corresponding to the rows of the matrices

## Examples

```
require(dplyr)
data(obama)

# Top words in the first Obama S.O.T.U., using all tokens
tfidf <- cnlp_utils_tfidf(obama)
vids <- order(tfidf[1,], decreasing = TRUE)[1:10]
colnames(tfidf)[vids]

# Top words, only using non-proper nouns
tfidf <- cnlp_get_token(obama) %>%
  filter(pos %in% c("NN", "NNS")) %>%
  cnlp_utils_tfidf()
vids <- order(tfidf[1,], decreasing = TRUE)[1:10]
colnames(tfidf)[vids]
```

---

# cnlp_write_conll *Returns a CoNLL-U Document*

---

## Description

Given an annotation object, this function returns a CoNLL-U document. The format of CoNLL-U is close to that of a deliminated file, but includes blank lines to signal breaks between sentences. We return a string object that can be saved to disk using the function `readLines`. Note that CoNLL-U does not have a way of distinguishing documents. Usually only one document is written to a single file. If you want this behavior, see the examples. Also note that this is a lossy procedure depending on the annotations available, saving just tokenization, lemmatization, part of speech tags, and dependencies.

## Usage

```
cnlp_write_conll(anno)
```

## Arguments

| | |
|---|---|
| anno | annotation object to convert |

## Value

an annotation object with a single document

### Author(s)

Taylor B. Arnold, <taylor.arnold@acm.org>

### Examples

```
## Not run:
for (i in cnlp_get_document(obama)$id) {
  anno <- extract_documents(obama, i)
  conll <- cnlp_write_conll(anno)
  writeLines(conll, sprintf("%02d.conll", i))
}

## End(Not run)
```

---

## cnlp_write_csv *Write annotation files to disk*

---

### Description

Takes an annotation object and stores it as a set of files in a local directory. These are stored as plain-text csv files with column headers. To save as a compressed format, instead directly call the function saveRDS.

### Usage

```
cnlp_write_csv(annotation, output_dir)
```

### Arguments

| | |
|---|---|
| annotation | annotation file being stored |
| output_dir | path to the directory where the files will be saved |

### Author(s)

Taylor B. Arnold, <taylor.arnold@acm.org>

### Examples

```
## Not run:
write_annotation(annotation, "/path/to/annotation")

## End(Not run)
```

## dep_frequency *Universal Dependency Frequencies*

### Description

The language-specific frequency of universal dependency codes as given in standard treebank corpora. Frequencies are multiplied by 100.

### References

http://universaldependencies.org/

## obama *Annotation of Barack Obama's State of the Union Addresses*

### Description

Parsed text from all eight State of the Union addresses given by Barack Obama.

### References

http://www.presidency.ucsb.edu/sou.php

## pos_frequency *Universal Part of Speech Code Frequencies*

### Description

The language-specific frequency of universal part of speech codes as given in standard treebank corpora. Frequencies are multiplied by 100.

### References

http://universaldependencies.org/

## print.annotation *Print a summary of an annotation object*

### Description

Print a summary of an annotation object

### Usage

```
## S3 method for class 'annotation'
print(x, ...)
```

### Arguments

| | |
|---|---|
| x | an annotation object |
| ... | other arguments. Currently unused. |

### Author(s)

Taylor B. Arnold, <taylor.arnold@acm.org>

## renamed *Renamed functions*

### Description

These functions have been renamed. For the most part they should now just be called with the prefix 'cnlp_'. See individual warning messages for the particular calling structure.

### Usage

```
get_coreference(...)

get_dependency(...)

get_document(...)

get_entity(...)

get_sentence(...)

get_tfidf(...)

get_token(...)
```

```
get_vector(...)

init_coreNLP(...)

init_spaCy(...)

init_tokenizers(...)

run_annotators(...)

tidy_pca(...)

to_CoNNL(...)

from_CoNNL(...)

write_annotation(...)

read_annotation(...)

download_core_nlp(...)

combine_documents(...)

extract_documents(...)
```

## Arguments

| | |
|---|---|
| ... | options passed to the newly named function |

---

# un *Universal Declaration of Human Rights*

---

## Description

Data frame containing the 30 Articles in the United Nations' Universal Declaration of Human Rights, ratified on 10 December 1948 in Paris, France.

## References

http://www.un.org/en/universal-declaration-human-rights/

word_frequency *Most frequent English words*

## Description

A dataset of the 150k most frequently used English words, extracted by Peter Norvig from the Google Web Trillion Word Corpus. Frequencies are multiplied by 100.

## References

http://norvig.com/ngrams/

# Index