A Guide to Using spacyr

Kenneth Benoit and Akitaka Matsuo

2020-03-04

Contents

Introduction

spacyr provides a convenient R wrapper around the Python spaCy (https://spacy.io) package. It offers easy access to the following functionality of spaCy:

- parsing texts into tokens or sentences;
- lemmatizing tokens;
- parsing dependencies (to identify the grammatical structure of the sentence); and
- identifying, extracting, or consolidating token sequences that form named entities or noun phrases.

It also allows a user to request additional token-level attributes directly from spaCy.

spacyr also takes care of the installation of not only spaCy but also Python itself, in a self-contained miniconda or virtualenv environment, and can install additional language models or upgrade spaCy as new models and versions become available.

Finally, **spacyr** works seamlessly with the **quanteda** (https://quanteda.io) package, although such use is optional.

Starting a spacyr session

spacyr works through the **reticulate** (https://github.com/rstudio/reticulate) package that allows R to harness the power of Python. To access the underlying Python functionality, **spacyr** must open a connection by being initialized within your R session.

We provide a function for this, `spacy_initialize()`, which attempts to make this process as painless as possible. When spaCy has been installed in a conda environment with `spacy_install()` (and see https://spacyr.quanteda.io (https://spacyr.quanteda.io) for detailed instructions on this setup), `spacy_initialize()` automatically detects it and initializes spaCy. If spaCy is installed in a normal environment (i.e. not in a condaenv or virtualenv), `spacy_initialize()` searches your system for Python executables, and testing which have spaCy installed.

For power users with a specialized setup of spaCy (i.e. users who have a conda environment already set up for spaCy), it is possible to specify which environment or python executable to be used through one of the following methods:

1. `condaenv` argument: supplying the name of conda environment
2. `virtualenv` argument: supplying the path to the python virtual environment
3. `python_executable` argument: supplying the path to the python

```
library("spacyr")
spacy_initialize(model = "en_core_web_sm")
## Found 'spacy_condaenv'. spacyr will use this environment
## successfully initialized (spaCy Version: 2.2.3, language model: en_core_web_sm)
## (python options: type = "condaenv", value = "spacy_condaenv")
```

Tokenizing and tagging texts

The `spacy_parse()` function is **spacyr**'s main workhorse. It calls spaCy both to tokenize and tag the texts. It provides two options for part of speech tagging, plus options to return word lemmas, recognize names entities or noun phrases recognition, and identify grammatical structures features by parsing syntactic dependencies. It returns a `data.frame` corresponding to the emerging *text interchange format* (https://github.com/ropensci/tif) for token data.frames.

The tokenization approach taken by spaCy is inclusive: it includes all tokens without restrictions, including punctuation characters and symbols.

Example:

```
txt <- c(d1 = "spaCy is great at fast natural language processing.",
         d2 = "Mr. Smith spent two years in North Carolina.")

# process documents and obtain a data.table
parsedtxt <- spacy_parse(txt)
parsedtxt
##    doc_id sentence_id token_id      token      lemma   pos   entity
## 1      d1           1        1      spaCy      spaCy PROPN
## 2      d1           1        2         is         be   AUX
## 3      d1           1        3      great      great   ADJ
## 4      d1           1        4         at         at   ADP
## 5      d1           1        5       fast       fast   ADJ
## 6      d1           1        6    natural    natural   ADJ
## 7      d1           1        7   language   language  NOUN
## 8      d1           1        8 processing processing  NOUN
## 9      d1           1        9          .          . PUNCT
## 10     d2           1        1        Mr.        Mr. PROPN
## 11     d2           1        2      Smith      Smith PROPN PERSON_B
## 12     d2           1        3      spent      spend  VERB
## 13     d2           1        4        two        two   NUM   DATE_B
## 14     d2           1        5      years       year  NOUN   DATE_I
## 15     d2           1        6         in         in   ADP
## 16     d2           1        7      North      North PROPN    GPE_B
## 17     d2           1        8   Carolina   Carolina PROPN    GPE_I
## 18     d2           1        9          .          . PUNCT
```

Two fields are available for part-of-speech tags. The `pos` field returned is the Universal tagset for parts-of-speech (http://universaldependencies.org/u/pos/all.html), a general scheme that most users will find serves their needs, and also that provides equivalencies across languages. **spacyr** also provides a more detailed tagset, defined in each spaCy language model. For English, this is the OntoNotes 5 version of the Penn Treebank tag set (https://spacy.io/docs/usage/pos-tagging#pos-tagging-english).

```
spacy_parse(txt, tag = TRUE, entity = FALSE, lemma = FALSE)
##    doc_id sentence_id token_id      token   pos tag
## 1      d1           1        1      spaCy PROPN NNP
## 2      d1           1        2         is   AUX VBZ
## 3      d1           1        3      great   ADJ  JJ
## 4      d1           1        4         at   ADP  IN
## 5      d1           1        5       fast   ADJ  JJ
## 6      d1           1        6    natural   ADJ  JJ
## 7      d1           1        7   language  NOUN  NN
## 8      d1           1        8 processing  NOUN  NN
## 9      d1           1        9          . PUNCT   .
## 10     d2           1        1        Mr. PROPN NNP
## 11     d2           1        2      Smith PROPN NNP
## 12     d2           1        3      spent  VERB VBD
## 13     d2           1        4        two   NUM  CD
## 14     d2           1        5      years  NOUN NNS
## 15     d2           1        6         in   ADP  IN
## 16     d2           1        7      North PROPN NNP
## 17     d2           1        8   Carolina PROPN NNP
## 18     d2           1        9          . PUNCT   .
```

The Penn Treebank is specific to English parts of speech. For other language models, the detailed tagset will be based on a different scheme. In the German language model, for instance, the universal tagset (`pos`) remains the same, but the detailed tagset (`tag`) is based on the TIGER Treebank (https://spacy.io/docs/usage/pos-tagging#pos-tagging-german) scheme. Full details are available from the spaCy models web page (https://spacy.io/models/).

Direct parsing of texts is also possible, using **spacy_tokenize()**. The options are designed to match those in the (https://quanteda.io/reference/tokens.html) `tokens()` function from the **quanteda** package. By default this returns a named list (where the document name is the list element name):

```
spacy_tokenize(txt)
## $d1
## [1] "spaCy"      "is"         "great"      "at"         "fast"      
## [6] "natural"    "language"   "processing" "."         
## 
## $d2
## [1] "Mr."      "Smith"    "spent"    "two"      "years"    "in"       "North"   
## [8] "Carolina" "."
```

but it can also output a data.frame:

```
spacy_tokenize(txt, remove_punct = TRUE, output = "data.frame") %>%
    tail()
##    doc_id    token
## 11     d2    spent
## 12     d2      two
## 13     d2    years
## 14     d2       in
## 15     d2    North
## 16     d2 Carolina
```

Extracting language properties from texts

Entity and noun phrase recognition

spacyr can extract entities, either named or "extended" (https://spacy.io/api/annotation#named-entities) from the output of `spacy_parse()`.

```
parsedtxt <- spacy_parse(txt, lemma = FALSE, entity = TRUE, nounphrase = TRUE)
entity_extract(parsedtxt)
##   doc_id sentence_id         entity entity_type
## 1     d2           1          Smith      PERSON
## 2     d2           1 North_Carolina         GPE
```

"Extended" entities including entities such as dates, events, and cardinal or ordinal quantities.

```
entity_extract(parsedtxt, type = "all")
##   doc_id sentence_id         entity entity_type
## 1     d2           1          Smith      PERSON
## 2     d2           1      two_years        DATE
## 3     d2           1 North_Carolina         GPE
```

One very useful feature is to use the consolidation functions to compound multi-word entities into single "tokens" (as they would in a language like German):

```
entity_consolidate(parsedtxt) %>%
    tail()
##    doc_id sentence_id token_id          token    pos entity_type
## 11     d2           1        2          Smith ENTITY      PERSON
## 12     d2           1        3          spent   VERB            
## 13     d2           1        4      two_years ENTITY        DATE
## 14     d2           1        5             in    ADP            
## 15     d2           1        6 North_Carolina ENTITY         GPE
## 16     d2           1        7              .  PUNCT            
```

In a similar manner to named entity extraction, **spacyr** can extract or concatenate [noun phrases* (or *noun chunks* (https://spacy.io/usage/linguistic-features#noun-chunks)).

```
nounphrase_extract(parsedtxt)
##   doc_id sentence_id                       nounphrase
## 1     d1           1                            spaCy
## 2     d1           1 fast_natural_language_processing
## 3     d2           1                        Mr._Smith
## 4     d2           1                        two_years
## 5     d2           1                   North_Carolina
```

Just as with entities, noun phrases can also be consolidated into single “tokens”:

```
nounphrase_consolidate(parsedtxt)
##    doc_id sentence_id token_id                            token        pos
## 1      d1           1        1                            spaCy nounphrase
## 2      d1           1        2                               is        AUX
## 3      d1           1        3                            great        ADJ
## 4      d1           1        4                               at        ADP
## 5      d1           1        5 fast_natural_language_processing nounphrase
## 6      d1           1        6                                .      PUNCT
## 7      d2           1        1                        Mr._Smith nounphrase
## 8      d2           1        2                            spent       VERB
## 9      d2           1        3                        two_years nounphrase
## 10     d2           1        4                               in        ADP
## 11     d2           1        5                   North_Carolina nounphrase
## 12     d2           1        6                                .      PUNCT
```

If a user’s only goal is entity or noun phrase extraction, then two functions make this easy without first parsing the entire text:

```
spacy_extract_entity(txt)
##   doc_id           text ent_type start_id length
## 1     d2          Smith   PERSON        2      1
## 2     d2      two years     DATE        4      2
## 3     d2 North Carolina      GPE        7      2
spacy_extract_nounphrases(txt)
##   doc_id                             text  root_text start_id root_id length
## 1     d1                            spaCy      spaCy        1       1      1
## 2     d1 fast natural language processing processing        5       8      4
## 3     d2                        Mr. Smith      Smith        1       2      2
## 4     d2                        two years      years        4       5      2
## 5     d2                   North Carolina   Carolina        7       8      2
```

Dependency parsing

Detailed parsing of syntactic dependencies is possible with the `dependency = TRUE` option:

```
spacy_parse(txt, dependency = TRUE, lemma = FALSE, pos = FALSE)
##    doc_id sentence_id token_id      token head_token_id  dep_rel   entity
## 1      d1           1        1      spaCy             2    nsubj
## 2      d1           1        2         is             2     ROOT
## 3      d1           1        3      great             2    acomp
## 4      d1           1        4         at             3     prep
## 5      d1           1        5       fast             8     amod
## 6      d1           1        6    natural             7     amod
## 7      d1           1        7   language             8 compound
## 8      d1           1        8 processing             4     pobj
## 9      d1           1        9          .             2    punct
## 10     d2           1        1        Mr.             2 compound
## 11     d2           1        2      Smith             3    nsubj PERSON_B
## 12     d2           1        3      spent             3     ROOT
## 13     d2           1        4        two             5   nummod   DATE_B
## 14     d2           1        5      years             3     dobj   DATE_I
## 15     d2           1        6         in             3     prep
## 16     d2           1        7      North             8 compound    GPE_B
## 17     d2           1        8   Carolina             6     pobj    GPE_I
## 18     d2           1        9          .             3    punct
```

Extracting additional token attributes

It is also possible to extract additional attributes of spaCy tokens (https://spacy.io/api/token#attributes) with the `additional_attributes` option. For example, detecting numbers and email addresses:

```
spacy_parse("I have six email addresses, including me@mymail.com.",
            additional_attributes = c("like_num", "like_email"),
            lemma = FALSE, pos = FALSE, entity = FALSE)
##   doc_id sentence_id token_id         token like_num like_email
## 1  text1           1        1             I    FALSE      FALSE
## 2  text1           1        2          have    FALSE      FALSE
## 3  text1           1        3           six     TRUE      FALSE
## 4  text1           1        4         email    FALSE      FALSE
## 5  text1           1        5     addresses    FALSE      FALSE
## 6  text1           1        6             ,    FALSE      FALSE
## 7  text1           1        7     including    FALSE      FALSE
## 8  text1           1        8 me@mymail.com    FALSE       TRUE
## 9  text1           1        9             .    FALSE      FALSE
```

Using other language models

By default, **spacyr** loads an English language model. You also can load spaCy's other language models (https://spacy.io/docs/usage/models) or use one of the language models with alpha support (https://spacy.io/docs/api/language-models#alpha-support) by specifying the `model` option when calling `spacy_initialize()`. We have successfully tested following language models with spaCy version 2.0.18.

Language	ModelName
German	`de`
Spanish	`es`
Portuguese	`pt`
French	`fr`
Italian	`it`
Dutch	`nl`

This is an example of parsing German texts.

```
## first finalize the spacy if it's loaded
spacy_finalize()
spacy_initialize(model = "de_core_news_sm")
## Python space is already attached.  If you want to switch to a different Python, please restart R.
## successfully initialized (spaCy Version: 2.2.3, language model: de_core_news_sm)
## (python options: type = "condaenv", value = "spacy_condaenv")

txt_german <- c(R = "R ist eine freie Programmiersprache für statistische Berechnungen und Grafiken. Sie wu
               python = "Python ist eine universelle, üblicherweise interpretierte höhere Programmiersprach
results_german <- spacy_parse(txt_german, dependency = FALSE, lemma = FALSE, tag = TRUE)
results_german
##    doc_id sentence_id token_id              token   pos   tag entity
## 1       R           1        1                  R PROPN    NE MISC_B
## 2       R           1        2                ist   AUX VAFIN
## 3       R           1        3               eine   DET   ART
## 4       R           1        4              freie   ADJ  ADJA
## 5       R           1        5 Programmiersprache  NOUN    NN
## 6       R           1        6                für   ADP  APPR
## 7       R           1        7       statistische   ADJ  ADJA
## 8       R           1        8       Berechnungen  NOUN    NN
## 9       R           1        9                und CCONJ   KON
## 10      R           1       10           Grafiken  NOUN    NN
## 11      R           1       11                  . PUNCT    $.
## 12      R           2        1                Sie  PRON  PPER
## 13      R           2        2              wurde   AUX VAFIN
## 14      R           2        3                von   ADP  APPR
## 15      R           2        4       Statistikern  NOUN    NN
## 16      R           2        5                für   ADP  APPR
## 17      R           2        6           Anwender  NOUN    NN
## 18      R           2        7                mit   ADP  APPR
## 19      R           2        8      statistischen   ADJ  ADJA
## 20      R           2        9           Aufgaben  NOUN    NN
## 21      R           2       10         entwickelt  VERB  VVPP
## 22      R           2       11                  . PUNCT    $.
## 23 python           1        1             Python  NOUN    NN MISC_B
## 24 python           1        2                ist   AUX VAFIN
## 25 python           1        3               eine   DET   ART
## 26 python           1        4        universelle   ADJ  ADJA
## 27 python           1        5                  , PUNCT    $,
## 28 python           1        6      üblicherweise   ADV   ADV
## 29 python           1        7     interpretierte   ADJ  ADJA
## 30 python           1        8             höhere   ADJ  ADJA
## 31 python           1        9 Programmiersprache  NOUN    NN
## 32 python           1       10                  . PUNCT    $.
## 33 python           2        1                Sie  PRON  PPER
## 34 python           2        2               will  VERB VMFIN
## 35 python           2        3              einen   DET   ART
## 36 python           2        4                gut   ADJ  ADJD
## 37 python           2        5           lesbaren   ADJ  ADJA
## 38 python           2        6                  , PUNCT    $,
## 39 python           2        7            knappen   ADJ  ADJA
## 40 python           2        8    Programmierstil  NOUN    NN
## 41 python           2        9            fördern  VERB VVINF
## 42 python           2       10                  . PUNCT    $.
spacy_finalize()
```

Note that the additional language models must first be installed in spaCy. When spaCy has been installed through `spacy_install()`, installation of additional language models is very simple. For example, the German language model can be installed (`spacy_download_langmodel('de')`). In other environments, you can install the model by entering `python -m spacy download de` in the console.

Integrating spacyr with other text analysis packages

With quanteda

The outputs and formats of **spacyr** are designed to integrate directly with the **quanteda** package.

For instance, many of its functions operate directly on **spacyr** objects, such as a parsed text.

```
require(quanteda, warn.conflicts = FALSE, quietly = TRUE)
docnames(parsedtxt)
## [1] "d1" "d2"
ndoc(parsedtxt)
## [1] 2
ntoken(parsedtxt)
## d1 d2
##  9  9
ntype(parsedtxt)
## d1 d2
##  9  9
```

Conversion of tokens is easily performed, and the tokenizers in **spacyr** tend to be smarter than the purely syntactic pattern-based parsers used by **quanteda**.

```
spacy_initialize(model = "en_core_web_sm")
## Python space is already attached.  If you want to switch to a different Python, please restart R.
## successfully initialized (spaCy Version: 2.2.3, language model: en_core_web_sm)
## (python options: type = "condaenv", value = "spacy_condaenv")
parsedtxt <- spacy_parse(txt, pos = TRUE, tag = TRUE)
as.tokens(parsedtxt)
## Tokens consisting of 2 documents.
## d1 :
## [1] "spaCy"      "is"         "great"      "at"         "fast"
## [6] "natural"    "language"   "processing" "."
##
## d2 :
## [1] "Mr."      "Smith"    "spent"    "two"      "years"    "in"       "North"
## [8] "Carolina" "."
as.tokens(parsedtxt, include_pos = "pos")
## Tokens consisting of 2 documents.
## d1 :
## [1] "spaCy/PROPN"     "is/AUX"          "great/ADJ"       "at/ADP"
## [5] "fast/ADJ"        "natural/ADJ"     "language/NOUN"   "processing/NOUN"
## [9] "./PUNCT"
##
## d2 :
## [1] "Mr./PROPN"      "Smith/PROPN"    "spent/VERB"     "two/NUM"
## [5] "years/NOUN"     "in/ADP"         "North/PROPN"    "Carolina/PROPN"
## [9] "./PUNCT"
as.tokens(parsedtxt, include_pos = "tag")
## Tokens consisting of 2 documents.
## d1 :
## [1] "spaCy/NNP"     "is/VBZ"        "great/JJ"      "at/IN"
## [5] "fast/JJ"       "natural/JJ"    "language/NN"   "processing/NN"
## [9] "./."
##
## d2 :
## [1] "Mr./NNP"      "Smith/NNP"    "spent/VBD"    "two/CD"       "years/NNS"
## [6] "in/IN"        "North/NNP"    "Carolina/NNP" "./."
```

The latter is useful for say, selecting only nouns, using “glob” pattern matching with **quanteda**’s `tokens_select()` function:

```
spacy_parse("The cat in the hat ate green eggs and ham.", pos = TRUE) %>%
    as.tokens(include_pos = "pos") %>%
    tokens_select(pattern = c("*/NOUN"))
## Tokens consisting of 1 document.
## text1 :
## [1] "cat/NOUN"  "hat/NOUN"  "eggs/NOUN" "ham/NOUN"
```

Direct conversion of just the spaCy-based tokens is also possible:

```
spacy_tokenize(txt) %>%
    as.tokens()
## Tokens consisting of 2 documents.
## d1 :
## [1] "spaCy"      "is"         "great"      "at"         "fast"      
## [6] "natural"    "language"   "processing" "."         
## 
## d2 :
## [1] "Mr."      "Smith"    "spent"    "two"      "years"    "in"       "North"   
## [8] "Carolina" "."
```

including for sentences, for which spaCy's recognition is very smart:

```
txt2 <- "A Ph.D. in Washington D.C.  Mr. Smith went to Washington."
spacy_tokenize(txt2, what = "sentence") %>%
    as.tokens()
## Tokens consisting of 1 document.
## text1 :
## [1] "A Ph.D. in Washington D.C."    "Mr. Smith went to Washington."
```

This also works well with entity recognition, e.g.

```
spacy_parse(txt, entity = TRUE) %>%
    entity_consolidate() %>%
    as.tokens() %>%
    head(1)
## Tokens consisting of 1 document.
## d1 :
## [1] "spaCy"      "is"         "great"      "at"         "fast"      
## [6] "natural"    "language"   "processing" "."
```

With tidytext

If you prefer a tidy approach to text analysis, **spacyr** works nicely because it returns parsed texts and (optionally) tokenized texts as data.frame-based objects.

```
if (!requireNamespace("tidytext", quietly = TRUE))
  install.packages("tidytext", repos = "https://cran.rstudio.com/")
library("tidytext")
unnest_tokens(parsedtxt, word, token) %>%
    dplyr::anti_join(stop_words)
## Joining, by = "word"
##   doc_id sentence_id token_id      lemma   pos tag   entity       word
## 1     d1           1        1      spaCy PROPN NNP               spacy
## 2     d1           1        5       fast   ADJ  JJ                fast
## 3     d1           1        6    natural   ADJ  JJ             natural
## 4     d1           1        7   language  NOUN  NN            language
## 5     d1           1        8 processing  NOUN  NN          processing
## 6     d2           1        2      Smith PROPN NNP PERSON_B      smith
## 7     d2           1        3      spend  VERB VBD               spent
## 8     d2           1        7      North PROPN NNP    GPE_B      north
## 9     d2           1        8   Carolina PROPN NNP    GPE_I   carolina
```

Part of speech filtering can then happen using **dplyr**:

```
spacy_parse("The cat in the hat ate green eggs and ham.", pos = TRUE) %>%
    unnest_tokens(word, token) %>%
    dplyr::filter(pos == "NOUN")
##   doc_id sentence_id token_id lemma  pos entity word
## 1  text1           1        2   cat NOUN         cat
## 2  text1           1        5   hat NOUN         hat
## 3  text1           1        8   egg NOUN        eggs
## 4  text1           1       10   ham NOUN         ham
```

Adherence to the "TIF" standard

spacyr's output was designed to conform to the Text Interchange Format (https://github.com/ropensci/tif), a cooperatively agreed standard structure for text package objects in R, such as corpus and token objects. `spacy_initialize()` can take a TIF corpus data.frame or character object as a valid input. Moreover, the data.frames returned by `spacy_parse()` and `entity_consolidate()` conform to the TIF tokens standard for data.frame tokens objects. This will make it easier to use with any text analysis package for R that works with TIF standard objects.

Finishing a session

When `spacy_initialize()` is executed, a background process of spaCy is attached in python space. This can take up a significant size of memory especially when a larger language model is used (e.g. en_core_web_lg (https://spacy.io/models/en#en_core_web_lg)). When you do not need the connection to spaCy any longer, you can remove the spaCy object by calling the `spacy_finalize()` function.

```
spacy_finalize()
```

By calling `spacy_initialize()` again, you can reattach the backend spaCy.

Developed by Kenneth Benoit, Akitaka Matsuo, European Research Council.

Site built with pkgdown (https://pkgdown.r-lib.org/) 1.4.1