Processing Don Quixote

NLP master course 2022-2023

Mariano Rico (mariano.rico@upm.es)

Document created on 2022-11-28

Table of contents

1 Read text from Internet and selection of text **2**

2 Basic checks **3**

3 Basic structuration **3**

4 Some cleaning **3**

5 Some numbers (chars, words, sentences) **4**

6 Sentence analysis: part of speech and more **7**

6.1 Using spaCy 7

6.2 Using udpipes 8

7 Querying the parse tree **10**

8 Relations beyond sentence level **13**

9 Finishing **13**

1 Read text from Internet and selection of text

Use the text of Don Quixote from https://www.gutenberg.org/files/2000/2000-0.txt. Notice that you can load this text in a browser, but typically you can see a maximum number of lines. That is, you cannot see the end of the Quixote.

Delete the header and the tail provided by gutenberg, knowing that both the last line of the header and the first of the tail contain the characters ***. Compute the number of lines.

```
urlQuijoteGutenber <-  "https://www.gutenberg.org/files/2000/2000-0.txt"
lines <- readLines(urlQuijoteGutenber,
                   encoding = "UTF-8") #It takes a few seconds
grep(pattern = "***", lines, fixed = TRUE) #Warning! Without fixed the regex is "\\*\\*\\*"
```

```
[1]    24 37704 37706
```

```
                                          #Result: 24 37704 37706 lines ids
linesQ <- lines[25:37703]
length(linesQ) #37,679
```

```
[1] 37679
```

However, a simple inspection of the first lines in `linesQ` shows us that there is a prologue. We are interested in the text of Cervantes so, we remove the prologue lines knowing that the Quixote begins with "En un lugar de".

```
grep(pattern = "En un lugar de",
     linesQ,
     fixed = TRUE) #Lines 1045 and 13513. The good one is the first one
```

```
[1]  1045 13513
```

```
linesQ <- linesQ[-c(1:1044)] #Remove the prologue
length(linesQ) #36,635
```

```
[1] 36635
```

```
linesQ[1:5]
```

```
[1] "En un lugar de la Mancha, de cuyo nombre no quiero acordarme, no ha mucho"
[2] "tiempo que vivía un hidalgo de los de lanza en astillero, adarga antigua,"
[3] "rocín flaco y galgo corredor. Una olla de algo más vaca que carnero,"
[4] "salpicón las más noches, duelos y quebrantos los sábados, lantejas los"
[5] "viernes, algún palomino de añadidura los domingos, consumían las tres"
```

We can join lines doing so:

```
paste(linesQ[1:5], collapse = " ")
```

```
[1] "En un lugar de la Mancha, de cuyo nombre no quiero acordarme, no ha mucho tiempo que vivía un hidal
```

2 Basic checks

It is important to ensure that we selected the right encoding. If you can read the text properly it is a good sign. However, a systematic solution like this would be better:

```
library(utf8)
#Chack encoding
linesQ[!utf8_valid(linesQ)] #character(0) ==> All lines are made of correct UTF-8 characters
```

```
character(0)
```

```
#Check character normalization. Specifically, the normalized composed form (NFC)
linesQ_NFC <- utf8_normalize(linesQ)
sum(linesQ_NFC != linesQ) #0 means all right. The text is in NFC.
```

```
[1] 0
```

3 Basic structuration

Obtain a vector (or a list) with the paragraphs in the text, considering paragraph as a **not empty** text block separated from another by two blank lines (three `\n`). Compute the number of paragraphs in Don Quixote.

```
stringQ <- paste(linesQ, collapse = "\n") #One big string
paragraphs <- unlist(strsplit(stringQ, "\\n\\n\\n"))#Warn! (1)strsplit returns a list,
                                                    #      (2)escape \n and
                                                    #      (3)by default, fixed=FALSE
                                                    #Using fixed=TRUE this should be
                                                    #     "\n\n\n", fixed = TRUE
parEmpty <- which(paragraphs == "") #No empty paragraphs
#paragraphs <- paragraphs[-parEmpty]
length(paragraphs) # 128
```

```
[1] 128
```

We can see the first 200 characteres of the first paragraph with

```
substring(paragraphs[1], 1, 200)
```

```
[1] "En un lugar de la Mancha, de cuyo nombre no quiero acordarme, no ha mucho\ntiempo que vivía un hida
```

4 Some cleaning

Although the original text was quite clean, we have adden some `\n` characters. Therefore, we can do some basic cleaning replacing the occurrences of the caracter `\n` by using the base function `gsub()`.
Any sequence of one or more characters `\n` will bw replaced by a unique space " ".

```
#Testing the regex
gsub("[\n]{1,}", " ", c(par1="with one \nbut also\n",
                        par2="with a seq of \n\nlike this"
                       )
    )
```

```
                    par1                        par2
     "with one  but also " "with a seq of  like this"
```

```
paragraphswoNL <- gsub("[\n]{1,}", " ", paragraphs) #wo = without
substring(paragraphswoNL[1], 1, 200)
```

```
[1] "En un lugar de la Mancha, de cuyo nombre no quiero acordarme, no ha mucho tiempo que vivía un hidal
```

Sometime we can get sequences of several spaces. We replace any occurrency of more two or more spaces by a unique space doing this:

```
paragraphs <- gsub("[ ]{2,}", " ", paragraphswoNL) #We reassign the varible paragraphs
substring(paragraphs[1], 1, 200)
```

```
[1] "En un lugar de la Mancha, de cuyo nombre no quiero acordarme, no ha mucho tiempo que vivía un hidal
```

5 Some numbers (chars, words, sentences)

We will statrt calculating the number of **non-empty** sentences using `spacy` from the vector of paragraphs.

HINT: The `spacyr` package uses spaCy (a Python library). Before using any functionality in `spacyr` you have to create a *Python environment*. Fortunatelly, the `spacyr` function `spacy_install()` does all this work. Once you have used the Python environment you should call `spacy_finalize()` to free Python resources (more than 1.5GB RAM).

HINT: By default, `spacyr` uses the English model. The Spanish model can be downloaded by using `spacy_download_langmodel('es')`. Currently downloads the `es_core_news_sm model` (`sm` comes from small). If you are insterested in downloading bigger models, follow this link (in Spanish, sorry).

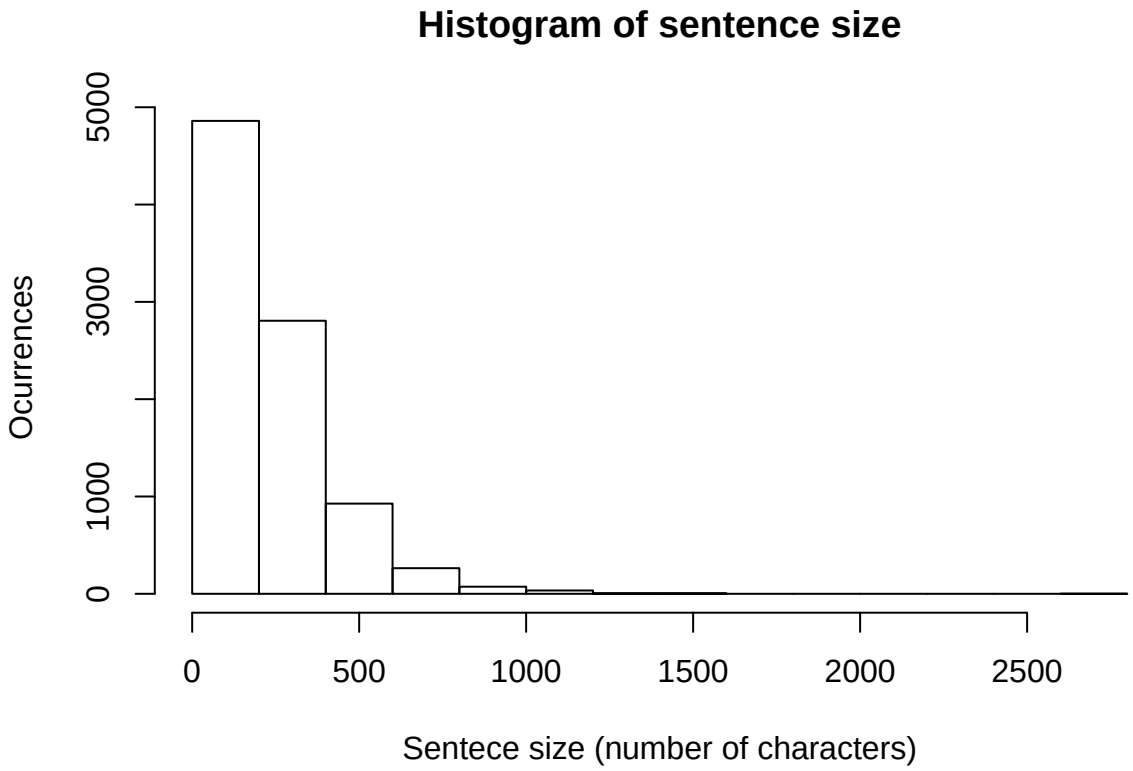
```
library(spacyr)
#Use spacy_install() if you have never used spacyr before. This will install a miniconda environment
#spacy_download_langmodel('es') #This downloads the model es_core_news_sm to disk
spacy_initialize(model = "es_core_news_sm") #Loads the Spanish model fron disk

#Gets sentences from paragraphs
phrases <- spacy_tokenize(paragraphs,      #If you use quanteda you can use
                                           #  corpus_reshape(corpus, to = "sentences"))
                                           #Taks a while.
                                           #Returns a list with 138 elements, each one
                                           #        is a string vector.
                          what="sentence" #By default remove_separators = TRUE
                                           #           (removes trailing spaces)
                         )

v_phrases <- unlist(phrases)
numphrases <- length(v_phrases) #8,975 sentences
sum(v_phrases=="") #1
```

```
[1] 1
```

```
v_phrases <- v_phrases[-which(v_phrases=="")] #8,974 sentences
```

What about the length of the sentences?

```
#A simple histogram will do fine
hist(nchar(v_phrases),
     main = "Histogram of sentence size",
     xlab = "Sentece size (number of characters)",
     ylab = "Ocurrences"
    )
```



We can compute the number of tokens using `spacy_tokenize`. Notice the number of options of this function. A token is not always just a word.

```
tokens <- spacy_tokenize(paragraphs
                         #Parameters asigned by default:
                           #remove_punct =      FALSE,  punt symbols are tokens
                           #remove_url =        FALSE,  url elements are tokens
                           #remove_numbers =    FALSE,  numbers are tokens
                           #remove_separators = TRUE,   spaces are NOT tokens
                           #remove_symbols =    FALSE,  symbols (like €) are tokens
                         )#Returns a list
v_tokens <- unlist(tokens)
v_tokens[1:10]
```

```
  text11   text12   text13   text14   text15   text16   text17   text18
```

```
   "En"     "un"  "lugar"     "de"     "la" "Mancha"      ","     "de"
 text19  text110
 "cuyo" "nombre"
```

```
length(v_tokens) #442,164 tokens (many repeated)
```

```
[1] 442164
```

```
length(unique(v_tokens)) #24,130 different (unique) tokens.
```

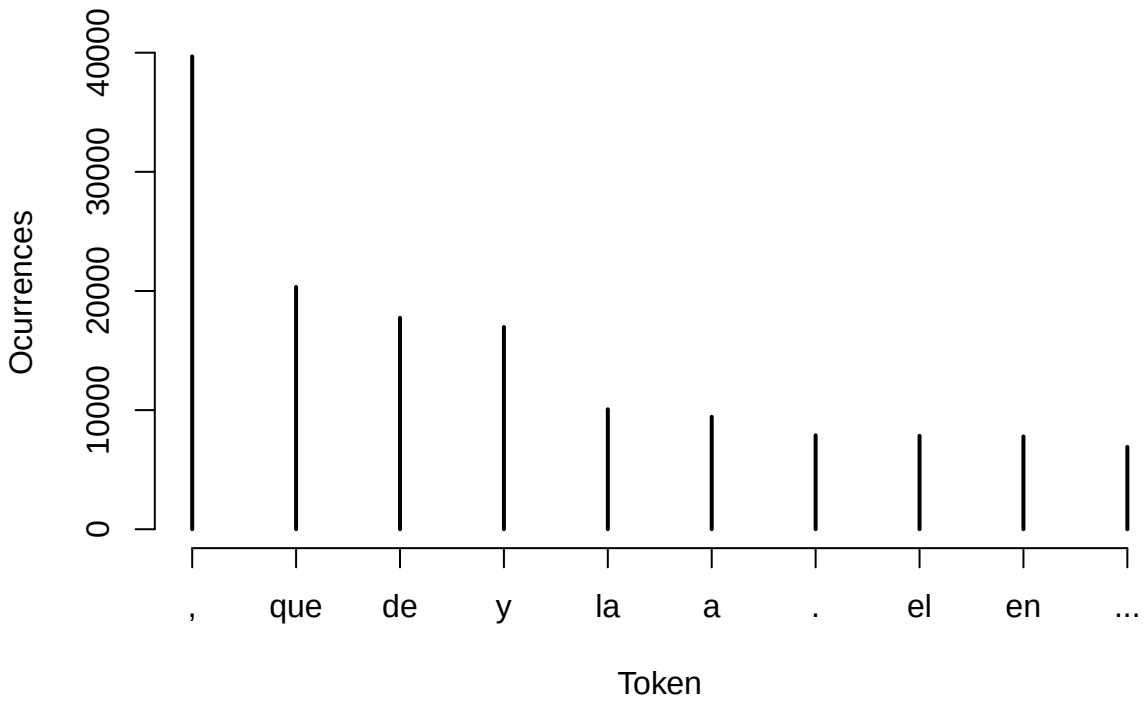
```
[1] 24130
```

Curious about the number of occurrencies of these tokens?

```
#As a list
head(sort(table(v_tokens), decreasing = TRUE), n = 25)
```

```
v_tokens
     ,    que     de      y     la      a      .     el     en      -
 39698  20340  17753  16977  10073   9440   7888   7843   7790   6915
    no      ;     se    los    con    por    las     le     lo     su
  5738   4745   4646   4634   4025   3727   3383   3378   3349   3304
   don    del     me   como Quijote
  2526   2429   2325   2223   2150
```

```
#As a simple plot
plot(head(sort(table(v_tokens), decreasing = TRUE), n = 10),
     xlab = "Token",
     ylab = "Ocurrences"
    )
```



6 Sentence analysis: part of speech and more

The part of speech (pos) is a heavy task that takes a long time to compute. Here we will test the pos functionality with two packages: `spacyr` and `udpipes`.

For this example we will compute the pos of the fist 100 sentences of Don Quixote.

6.1 Using spaCy

SpaCy is good identifiying sentences in Spanish, better than other packages such as `udpipes` or `quanteda`. Also is good doing Part of Speech (morphosyntactic analysis) and sophisticated tasks such as named entity recognition (NER), noun phrase detection, and dependencies. All this information is located in the `spacy_parse()` function.

For this example we will compute the pos of the first 100 sentences of Don Quixote.

```
#begin <- Sys.time()
#spacy_parse() doesn't like duplicated names (and we have sum(duplicated(names(v_phrases))) = 477)
#Therefore we remove the names of the text strings before using spacy_parse().
#names(v_phrases) <- NULL
# res <- spacy_parse(v_phrases, #If you use phrases, spacy will take just 50 secs (in my machine)
#                               #If you use paragraphs, spacy will take just 1.22 mins (in my machine)
#                               # but it will finish in token 441,979 (that is, before the end)
#                               # without any error. spacyr is not very kind :-S
#                                     #Default params
#                                         #pos =        TRUE,
#                                         #tag =        FALSE,   #Nothing extra for Spanish
#                                         #lemma =      TRUE,
#                                         #entity =     TRUE,
#                     dependency = TRUE,  #dependency = FALSE,
#                     nounphrase = TRUE   #nounphrase = FALSE
#                   ) #returns a list of dataframes
# Sys.time()-begin
# tic <- Sys.time()

#We will use lapply to send to space_parse() sentences one by one
res <- lapply(v_phrases[1:100],
              spacy_parse, #This is the function to apply to every element in v_phrases
              dependency = TRUE, nounphrase = TRUE #These are the arguments of the function
             ) #Returns a list. Each list element is a data frame with 11 columns (in most cases!!)

# Sys.time()-tic
# tic <- Sys.time()
# #Check if thare is a df without 11 cols
# ncols <- lapply(res, ncol)
# sum(ncols !=11) #There are 22 !! :-O
# which(ncols !=11) #253, 2246, 2885, 2932, 4338, 4343, 4356, 4524, 4840, 4844, 5828, 5992, 6488, 6502
#                   #6504, 6592, 6993, 7345, 7405, 7648, 8060, 8255
#                   #These sentences do not have cols nounphrase nor whitespace. Only have 9 cols
# table(unlist(ncols)) #All the cases have a 9 col data frames
# #A solution would be to create, for these 11 dataframes, empty cols.
#
# Sys.time()-tic
#
```

```r
#Maka big df
 df <- res[[1]] #A data frame with the first resuls
 for (i in 2:length(res)){ #Attention! The loop starts from 2
   df <- rbind(df, res[[i]])
 }
# Sys.time()-tic
# #As this takes a while, I save the result
# saveRDS(df, file="spacy_parse_Quixote.rds")

#Shows the first 20 tokens.
library(kableExtra) #Styling the kable output to show very width data frames
kable_styling(kable(df[1:20, c(3:ncol(df))]),  #The first 2 cols UNSHOWN are doc_id and sentence_id
                    font_size = 7
                   )
```

token_id	token	lemma	pos	head_token_id	dep_rel	entity	nounphrase	whitespace
1	En	en	ADP	3	case			TRUE
2	un	uno	DET	3	det		beg	TRUE
3	lugar	lugar	NOUN	18	obl		end_root	TRUE
4	de	de	ADP	6	case			TRUE
5	la	el	DET	6	det	LOC_B	beg	TRUE
6	Mancha	Mancha	PROPN	3	nmod	LOC_I	end_root	FALSE
7	,	,	PUNCT	12	punct			TRUE
8	de	de	ADP	10	case			TRUE
9	cuyo	cuyo	PRON	10	nmod		beg_root	TRUE
10	nombre	nombre	NOUN	12	obj		beg_root	TRUE
11	no	no	ADV	12	advmod			TRUE
12	quiero	querer	VERB	6	acl			TRUE
13	acordarme	acordar yo	VERB	12	xcomp			FALSE
14	,	,	PUNCT	12	punct			TRUE
15	no	no	ADV	18	advmod			TRUE
16	ha	haber	AUX	18	cop			TRUE
17	mucho	mucho	DET	18	det		beg	TRUE
18	tiempo	tiempo	NOUN	18	ROOT		end_root	TRUE
19	que	que	SCONJ	20	mark			TRUE
20	vivía	vivir	VERB	18	acl			TRUE

6.2 Using udpipes

The package **udpipes** has most functionalities of **spacyr** (for different languages, Spanish included) with the exception of name entity recognition. However, **spacyr** is around 6 times faster than udpipes. You can see the comparation here (sorry, in Spanish). Specifically **udpipes** can do from raw text: tokenization, parts of speech tagging, lemmatization and dependency parsing. Also has usefull functions like: collocations, token co-occurrence, document term matrix handling, term frequency and inverse document frequency calculations, handling of multi-word expressions, noun phrase extraction, handling of syntactical patterns, among other.

Another usefull funcionality in **udpipes** is that it can save/load the annotations in coNLL format, a very popular annotation format.

```r
library(udpipe)
model_file <- 'spanish-ancora-ud-2.5-191206.udpipe'
if(!file.exists(model_file)){
  model <- udpipe_download_model(language = "spanish-ancora") #Another alternative: "spanish-gsd"
  udmodel_es <- udpipe_load_model(file = model$file_model)
}else{
  udmodel_es <- udpipe_load_model(file = model_file)
```

```
}

tic <- Sys.time()
anno <- udpipe_annotate(udmodel_es,
                        x = v_phrases[1:100],
                        parallel.cores = 10 #Check your system!!
                       )
df <- as.data.frame(anno)
Sys.time()-tic
```

```
Time difference of 5.835727 secs
```

```
## Pay attention
#anno is a list containing 3 things (last 2 where lost converting to data frame):
# 1) x: the character vector with text.
# 2) conllu: annnotation in CONLL-U format
# 3) error: A vector with the same length of x containing possible errors when annotating x

#Write the result as a coNLL file
cat(anno$conllu, file = "udpipes_es_Quixote.conllu")
#You can read this file with udpipe_read_conllu()

#Show the annotations of the first 20 tokens
#As df has 14 columns, we show them in two tables
library(kableExtra) #Styling the kable output to show very width data frames
kable_styling(kable(df[1:20, c(5:9)]),  #The first 4 cols UNSHOWN are
                                        #doc_id, paragraph_id, sentence_id and sentence
              font_size = 7
             )
```

token_id	token	lemma	upos	xpos
1	En	en	ADP	ADP
2	un	uno	DET	DET
3	lugar	lugar	NOUN	NOUN
4	de	de	ADP	ADP
5	la	el	DET	DET
6	Mancha	Mancha	PROPN	PROPN
7	,	,	PUNCT	PUNCT
8	de	de	ADP	ADP
9	cuyo	cuyo	PRON	PRON
10	nombre	nombre	NOUN	NOUN
11	no	no	ADV	ADV
12	quiero	querer	VERB	VERB
13-14	acordarme	NA	NA	NA
13	acordar	acordar	VERB	VERB
14	me	yo	PRON	PRON
15	,	,	PUNCT	PUNCT
16	no	no	ADV	ADV
17	ha	haber	AUX	AUX
18	mucho	mucho	DET	DET
19	tiempo	tiempo	NOUN	NOUN

```
kable_styling(kable(df[1:20, c(10:14)]),  #Remaining cols
              font_size = 7
             )
```

feats	head_token_id	dep_rel	deps	misc
AdpType=Prep	3	case	NA	NA
Definite=Ind|Gender=Masc|Number=Sing|PronType=Art	3	det	NA	NA
Gender=Masc|Number=Sing	17	obl	NA	NA
AdpType=Prep	6	case	NA	NA
Definite=Def|Gender=Fem|Number=Sing|PronType=Art	6	det	NA	NA
NA	3	nmod	NA	SpaceAfter=No
PunctType=Comm	12	punct	NA	NA
AdpType=Prep	10	case	NA	NA
Gender=Masc|Number=Sing|Poss=Yes|PronType=Int,Rel	10	nmod	NA	NA
Gender=Masc|Number=Sing	12	obl	NA	NA
Polarity=Neg	12	advmod	NA	NA
Mood=Ind|Number=Sing|Person=1|Tense=Pres|VerbForm=Fin	6	acl	NA	NA
NA	NA	NA	NA	SpaceAfter=No
VerbForm=Inf	12	xcomp	NA	NA
Case=Acc,Dat|Number=Sing|Person=1|PrepCase=Npr|PronType=Prs	13	obj	NA	NA
PunctType=Comm	12	punct	NA	NA
Polarity=Neg	17	advmod	NA	NA
Mood=Ind|Number=Sing|Person=3|Tense=Pres|VerbForm=Fin	0	root	NA	NA
Gender=Masc|Number=Sing|NumType=Card|PronType=Ind	19	det	NA	NA
Gender=Masc|Number=Sing	17	obj	NA	NA

7 Querying the parse tree

The package (rsyntax)[https://cran.r-project.org/web/packages/rsyntax] allows you to make queries to the parse tree. Specifically, from an annotations data frame (from spacyr or udpipe, among others), this library can select rows in the data frame (i.e,. tokens) that follow some dependency tree pattern.

The dependency tree pattern can include POS elements (like VERB, or PROPN), relations like `subj` or `obj`, and hierarchical relations like `child`.

There you have an example (extracted from the package documentation):

```
library(udpipe)
library(rsyntax)

tokens <- udpipe('Mary Jane loves John Smith, and Mary is loved by John',
                 'english') #English sentence

kable_styling(kable(tokens[, c("token_id", "token", "lemma", "upos", "head_token_id", "dep_rel")]),
              font_size = 7
             )
```

token_id	token	lemma	upos	head_token_id	dep_rel
1	Mary	Mary	PROPN	3	nsubj
2	Jane	Jane	PROPN	1	flat
3	loves	love	VERB	0	root
4	John	John	PROPN	3	obj
5	Smith	Smith	PROPN	4	flat
6	,	,	PUNCT	10	punct
7	and	and	CCONJ	10	cc
8	Mary	Mary	PROPN	10	nsubj:pass
9	is	be	AUX	10	aux:pass
10	loved	love	VERB	3	conj
11	by	by	ADP	12	case
12	John	John	PROPN	10	obl

```
plot_tree(tokens,
          token, lemma, upos #will use the $token and $pos columns in tokens
          )
```

The previous code produces figure 1.

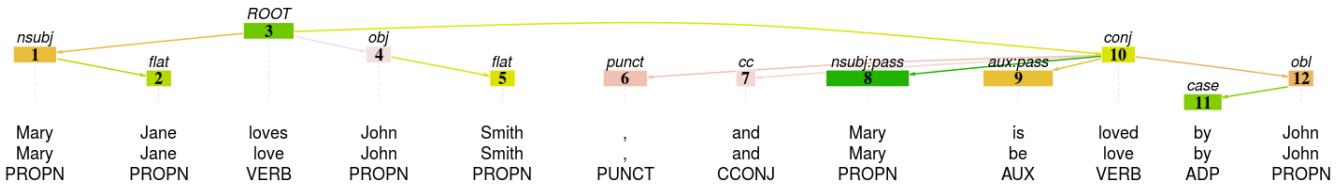


Figure 1: Parse tree created with package rsyntax.

You can create queries using `tquery()` like these:

```
#Direct relation. Find all tokens where upos is 'VERB', and that have a child with the
#  relation 'nsubj' AND a child with the relation 'obj'.
#  If this condition is met, give these tokens the labels 'verb', 'subject' and 'object'.
#  These new labels will be added to the data
direct <- tquery(label = 'verb', upos = 'VERB',
                 children(label = 'subject', relation = 'nsubj'),
                 children(label = 'object', relation = 'obj'))

#Notice the nice viualization for the query
direct
```

```
  n              verb           upos=VERB
    c            subject        relation=nsubj
      c          subject_FILL   req=F, depth=Inf
    c            object         relation=obj
      c          object_FILL    req=F, depth=Inf
    c            verb_FILL      req=F, depth=Inf
```

Relations (like 'nsubj' or 'obj') are defined in the udpipe documentation.

We can aggregate the elements of a multiword expression (MWE) with `fill()` (see documentation for a detailed explanation). The function `annotate_tqueries` applies the query (generating new columns in the dataframe):

```
fill_mwe <- fill(relation = c('flat','fixed','compound'),
                 connected=T)

direct_wme <- tquery(label = 'verb', upos = 'VERB', fill=F,
                     children(label = 'subject', relation = 'nsubj',
                              fill_mwe),
                     children(label = 'object', relation = 'obj',
                              fill_mwe))

tokens_direct_wme <- annotate_tqueries(tokens,
                                       'clause', #Name of the new columns name in the data frame
                                       direct,
                                       overwrite = TRUE) #Otherwise we will not update col 'clause'

kable_styling(kable(tokens_direct_wme[,
```

```
                        c("token_id", "token", "lemma", "upos",
                          #"head_token_id", "dep_rel"          #These are gone!!
                          "clause", "clause_id", "clause_fill" #These are new!!
                          )]),
            font_size = 7)
```

token_id	token	lemma	upos	clause	clause_id	clause_fill
1	Mary	Mary	PROPN	subject	doc1.1.3	0
2	Jane	Jane	PROPN	subject	doc1.1.3	1
3	loves	love	VERB	verb	doc1.1.3	0
4	John	John	PROPN	object	doc1.1.3	0
5	Smith	Smith	PROPN	object	doc1.1.3	1
6	,	,	PUNCT	verb	doc1.1.3	2
7	and	and	CCONJ	verb	doc1.1.3	2
8	Mary	Mary	PROPN	verb	doc1.1.3	2
9	is	be	AUX	verb	doc1.1.3	2
10	loved	love	VERB	verb	doc1.1.3	1
11	by	by	ADP	verb	doc1.1.3	3
12	John	John	PROPN	verb	doc1.1.3	2

You can plot the result with `plot_tree()`:

```
plot_tree(tokens_direct_wme, token, lemma, upos,
          annotation='clause') #Pay attention!
```

The previous code produces figure 2.

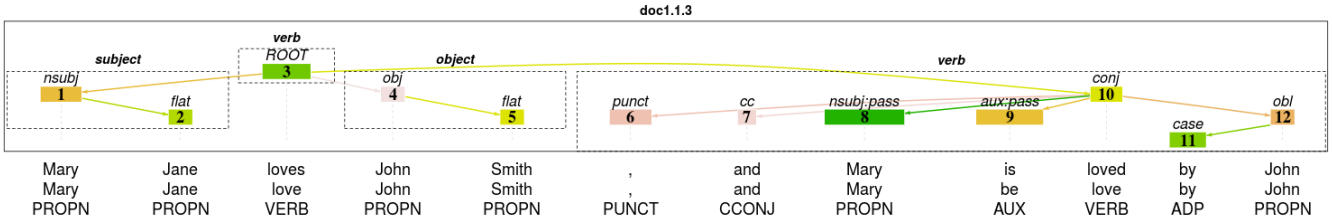


Figure 2: New boxes for `subject`, `verb` and `object`. Notice that subject and object are MWE.

We are also interested in the passive form at the end of the sentence ("Mary is loved by John"). This passive form has different relations that the direct form.

In the direct form we had the relation `nsubj` between the verb and the subject. In the passive form the relation is `obl` (as you can see in figure 2).

In the direct form we had the relation `obj` between the verb and the object. In the passive form the relation is `nsubj:pass` (as you can see in figure 2).

```
passive <- tquery(label = 'verb', upos = 'VERB', fill=FALSE,
                  children(label = 'subject', relation = 'obl',        fill_mwe),
                  children(label = 'object',  relation = 'nsubj:pass', fill_mwe))
tokens_both <- annotate_tqueries(tokens, 'clause',
                                 dir = direct,  #New prefix `dir` in col `clause_id`
                                 pas = passive, #New prefix `pas` in col `clause_id`
                                 overwrite = TRUE)
```

```
kable_styling(kable(tokens_both[,
                            c("token_id", "token", "lemma", "upos",
                              #"head_token_id", "dep_rel"          #These are gone!!
                              "clause", "clause_id", "clause_fill" #These are new!!
                              )]),
              font_size = 7)
```

token_id	token	lemma	upos	clause	clause_id	clause_fill
1	Mary	Mary	PROPN	subject	dir#doc1.1.3	0
2	Jane	Jane	PROPN	subject	dir#doc1.1.3	1
3	loves	love	VERB	verb	dir#doc1.1.3	0
4	John	John	PROPN	object	dir#doc1.1.3	0
5	Smith	Smith	PROPN	object	dir#doc1.1.3	1
6	,	,	PUNCT	NA	NA	NA
7	and	and	CCONJ	NA	NA	NA
8	Mary	Mary	PROPN	object	pas#doc1.1.10	0
9	is	be	AUX	NA	NA	NA
10	loved	love	VERB	verb	pas#doc1.1.10	0
11	by	by	ADP	NA	NA	NA
12	John	John	PROPN	subject	pas#doc1.1.10	0

```
plot_tree(tokens_both, token, lemma, upos,
          annotation='clause') #Pay attention!
```

The previous code produces figure 3.

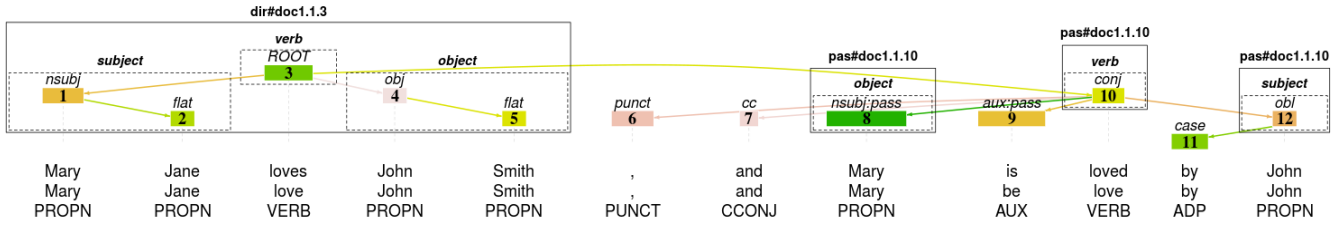


Figure 3: Query for direct and passive forms. For both cases we identify the subject and the object.

8 Relations beyond sentence level

If you want to detect relations beyond sentence level, you have to use **coreferences**. For example, in the sentence "he did […]", the token "he" refers to a previous entity. This relation is a coreference. The package `coreNLP` (a wrapper around the Java library created by Stanford University) allows you to compute coreferences.

9 Finishing

Do not forget to free Python resources used by `spacyr`.

```
spacy_finalize() #Do not forget this
```

In order to reproduce these results here is the session info:

```
sessionInfo()
```

```
R version 3.6.3 (2020-02-29)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: Ubuntu 16.04.7 LTS

Matrix products: default
BLAS:   /usr/lib/libblas/libblas.so.3.6.0
LAPACK: /usr/lib/lapack/liblapack.so.3.6.0

locale:
 [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
 [3] LC_TIME=en_US.UTF-8        LC_COLLATE=en_US.UTF-8
 [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
 [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
 [9] LC_ADDRESS=C               LC_TELEPHONE=C
[11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods   base

other attached packages:
[1] rsyntax_0.1.3    udpipe_0.8.8     kableExtra_1.1.0 spacyr_1.2.1
[5] utf8_1.1.4

loaded via a namespace (and not attached):
 [1] Rcpp_1.0.7        compiler_3.6.3    pillar_1.6.4      base64enc_0.1-3
 [5] tools_3.6.3       digest_0.6.27     viridisLite_0.3.0 jsonlite_1.6.1
 [9] evaluate_0.14     tibble_3.0.1      lifecycle_1.0.1   lattice_0.20-41
[13] png_0.1-7         pkgconfig_2.0.3   rlang_0.4.11      igraph_1.2.5
[17] Matrix_1.3-4      rstudioapi_0.11   yaml_2.2.1        xfun_0.13
[21] xml2_1.3.2        httr_1.4.1        stringr_1.4.0     knitr_1.28
[25] vctrs_0.3.8       rappdirs_0.3.1    hms_0.5.3         tidyselect_1.1.0
[29] webshot_0.5.1     grid_3.6.3        reticulate_1.15   glue_1.4.2
[33] data.table_1.12.8 R6_2.4.1          fansi_0.4.1       rmarkdown_2.1
[37] purrr_0.3.4       readr_1.3.1       magrittr_2.0.1    scales_1.0.0
[41] htmltools_0.4.0   ellipsis_0.3.2    rvest_0.3.5       colorspace_1.4-1
[45] stringi_1.7.5     munsell_0.5.0     crayon_1.3.4
```