Analizing Don Quixote and some other things

NLP master course 2021-2022

Mariano Rico (mariano.rico@upm.es)

Document created on 2022-12-13

Table of contents

1 Subword tokenization

Here we will compare two libraries (packages in R jargon): `tokenizers.bpe` and `sentencepiece`.

1.1 Getting the data

We will use the Don Quixote text again. Here (file Qcaps.rds) you have an R object serialized. This object is a list of strings, in which each string is a chapter from Cervantes' novel.

You should know that the novel has two parts. The first part goes from `Capítulo primero` to `Capítulo XXXVII` (both included). The second part goes from `Capítulo XXXVIII` to the end.

In the list `Qcaps`, the first part will go from `caps[[53]]` to `caps[[89]]` (both included). The second part will be from `caps[[90]]` to `caps[[126]]` (both included).

```
caps <- readRDS(file="Qcaps.rds")
text_part1 <- paste(unlist(caps[53:89]),  collapse="\n")
text_part2 <- paste(unlist(caps[90:126]), collapse="\n")
```

1.2 Creating a BPE model

We will use the first part of Don Quixote to train a BPE model, and we will apply the model to the second part.

Using `tokenizers.bpe`

```
library(tokenizers.bpe)

#I can't use a single line with all the text (text_part1), but I can use a vector of chapters
model <- bpe(unlist(caps[53:89]))

#We apply the model to the second part of the text (here we can use a single string)
subtoks2 <- bpe_encode(model, x = text_part2, type = "subwords")
head(unlist(subtoks2), n=20)
```

```
 [1] " Capítulo" " XXX"      "V"         "III."       " Donde"     " se"
 [7] " cuenta"   " la"       " que"      " dio"       " de"        " su"
[13] " mala"     " and"      "anza"      " la"        " dueña"     " Dolor"
[19] "ida"       " De"
```

Notice that the character for *cutted word* is not displayed correctly in this document, but it should be displayed correctly on the console. That character is not an underscore `"_"`, but a special character (U+2581) similar to a bold underline. Of course, `"\ U2581"!="_"`.

We can make a function to replace the character `"\U2581"` by `"_"` and thus display the result with more easily printable characters:

```
niceSubwords <- function(strings){
  gsub("\U2581", "_", strings)
}

niceSubwords(head(unlist(subtoks2), n=20))
```

```
 [1] "_Capítulo" "_XXX"      "V"         "III."      "_Donde"    "_se"
 [7] "_cuenta"   "_la"       "_que"      "_dio"      "_de"       "_su"
[13] "_mala"     "_and"      "anza"      "_la"       "_dueña"    "_Dolor"
[19] "ida"       "_De"
```

If you have a look at the documentation (type `?bpe` in the Console tab) you will see that you have used these default parameters: `coverage = 0.9999` and `vocab_size = 5000`.

Using `sentencepiece`

Apparently, the only difference is that the text must be provided in a file. However this package is much more *picky*: **BEWARE!** If we put the chapters text, it is too much text per line, resulting in a file not found error (which does not make any sense).

Therefore, we will provide the text by sentences (one sentence per line).

```
library(sentencepiece)

#We will use the spanish language model to get a good identification of sentences.
library(spacyr)
#spacy_install(prompt=FALSE)
#spacy_download_langmodel('es')
spacy_initialize(model = "es_core_news_sm") #Downloaded by spacy_download_langmodel('es')
sentences_part1 <- spacy_tokenize(text_part1, what="sentence") #Returns a list
v_sentences_part1 <- unlist(sentences_part1) #We get 2401 sentences

#Write the sentences in a file
train_file <- "Qsentencepiece.BPE.part1.txt"
writeLines(text = v_sentences_part1, con = train_file)

#Create a BPE model
model <- sentencepiece(train_file,          #File with sentences
                       type = "bpe",        #A BPE model. There are other models, like unigram
                       coverage = 0.9999,   #Default value 0.9999
                       vocab_size = 5000,   #Default value 5000
                       #threads = 1,        #By defaul it is 1. However, tokenizers.bpe uses all availa
                       model_dir = getwd(), #Current directory. Creates two files:
                                            #     sentencepiece.model and sentencepiece.vocab.
                       verbose = FALSE)     #Useful for debugging

#We apply the model to the second part of the text (here we can use a single string with whole text)
subtoks2_sentencepiece <- sentencepiece_encode(model, x = text_part2, type = "subwords")
niceSubwords(head(unlist(subtoks2_sentencepiece), n=20)) #tokenization made by sentencepiece
```

```
 [1] "_Capítulo" "_XXX"      "V"         "III"       "."         "_Donde"
 [7] "_se"       "_cuenta"   "_la"       "_que"      "_dio"      "_de"
[13] "_su"       "_mala"     "_and"      "anza"      "_la"       "_dueña"
[19] "_Dolorida" "_De"
```

The tokenizations achieved by the two libraries are different. Notice that, although none of these libraries consider any language model, in the case of sentencepiece we provided a more structured text (sentences, using the `spacyr` spanish model). Obviously we also can use this structured text in `tokenizers.bpe`:

```r
model <- bpe(v_sentences_part1)
subtoks2_alt <- bpe_encode(model, x = text_part2, type = "subwords")
niceSubwords(head(unlist(subtoks2_alt), n=20))
```

```
 [1] "_Capítulo" "_XXX"      "V"         "III."      "_Donde"    "_se"
 [7] "_cuenta"   "_la"       "_que"      "_dio"      "_de"       "_su"
[13] "_mala"     "_and"      "anza"      "_la"       "_dueña"    "_Dolor"
[19] "ida"       "_De"
```

We can see that the result is the same than before. We conclude that doesn't matter the way of providing texts (chapters or sentences) to `tokenizers.bpe`.

We can provide a better visualization with this:

```r
v <- unlist(subtoks2_alt)
strwrap(niceSubwords(substring(paste(v, collapse = " "), 1, 200)), exdent = 2)
```

```
[1] "_Capítulo _XXX V III. _Donde _se _cuenta _la _que _dio _de _su _mala"
[2] "  _and anza _la _dueña _Dolor ida _De trás _de _los _trist es _mús icos"
[3] "  _comenzaron _a _entrar _por _el _jar d ín _adelante _hasta _c"
```

```r
#The pipe quivalent is this
library(magrittr) #although it is included by many pckages, this is the original
paste(v, collapse = " ") %>%   #Creates a string with the tokens
  substring(1, 200) %>%        #Keep the first 200 characters
    niceSubwords() %>%         #Aplies the funcion that provides printable underscores
      strwrap(exdent = 2)      #A nice resulting string with indent
```

```
[1] "_Capítulo _XXX V III. _Donde _se _cuenta _la _que _dio _de _su _mala"
[2] "  _and anza _la _dueña _Dolor ida _De trás _de _los _trist es _mús icos"
[3] "  _comenzaron _a _entrar _por _el _jar d ín _adelante _hasta _c"
```

2 Distance between texts

With the TF-IDF matrix we have an *extensive vectorization* (mostly empty) for each document (remember that "document" can be a phrase, a chapter or the entire Don Quixote), but allows us to calculate distances between documents by calculating the distances (angles) between the vectors.

Using as documents the chapters of Don Quixote, we will compute the TF-IDF matrix and we will use the `hclust ()` function to show up which chapters are more similar to each other. We will create dendrograms with 4 groups ($k = 4$) and 10 groups ($k = 10$).

```r
#Creates a quanteda corpus
library(quanteda)
texts_caps <- unlist(caps)
names(texts_caps) <- paste("Chap.", 1:length(texts_caps)) #assigns a name to each string
corpus_capsQ <- corpus(texts_caps)
docvars(corpus_capsQ, field="Chapter") <- 1:length(texts_caps) #docvar with chapter number
corpus_capsQ
```

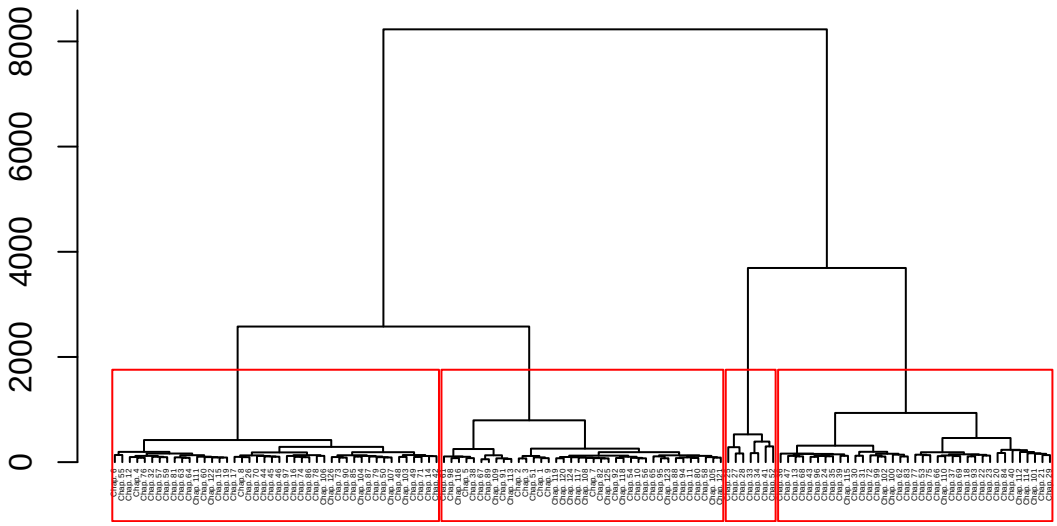
```
Corpus consisting of 126 documents and 1 docvar.
Chap. 1 :
"Capítulo primero. Que trata de la condición y ejercicio del ..."

Chap. 2 :
"Capítulo II. Que trata de la primera salida que de su tierra..."

Chap. 3 :
"Capítulo III. Donde se cuenta la graciosa manera que tuvo do..."

Chap. 4 :
"Capítulo IV. De lo que le sucedió a nuestro caballero cuando..."

Chap. 5 :
"Capítulo V. Donde se prosigue la narración de la desgracia d..."

Chap. 6 :
"Capítulo VI. Del donoso y grande escrutinio que el cura y el..."

[ reached max_ndoc ... 120 more documents ]
```

```r
#Creates the dfm (document-feature matrix)
dfm_capsQ <- dfm(tokens(corpus_capsQ),
                 #Default values:
                 # tolower = TRUE           #Convers to lowercase
                 # remove_padding  = FALSE  #Does padding (fills with blanks)
                )
#Does a dendrogram
distMatrix <-dist(as.matrix(dfm_capsQ),
                  method="euclidean")
groups <-hclust(distMatrix , method="ward.D")
```

2.1 Dendrograms

Draw the dendrogram with 4 aggrupations:

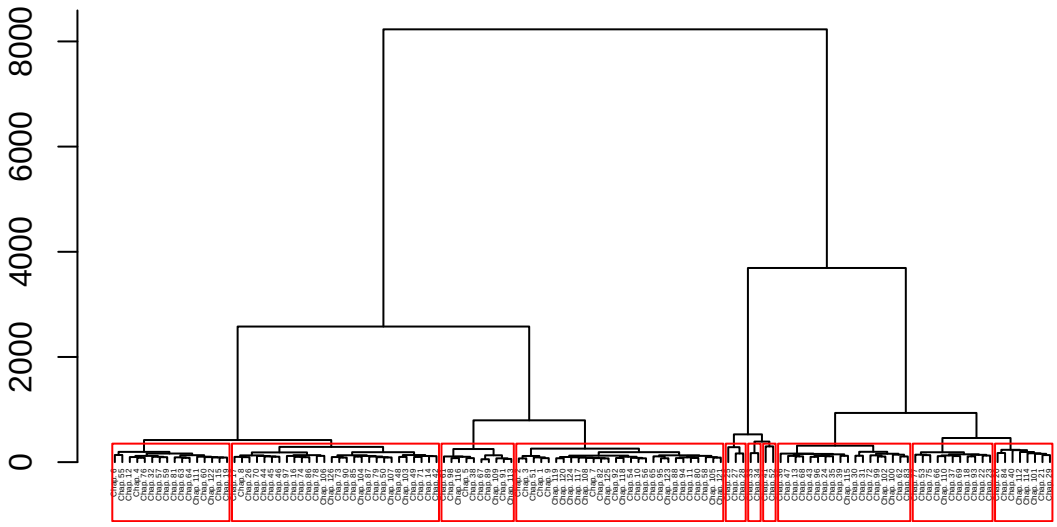
```r
plot(groups,
     cex =0.25, #Size of labels
     hang= -1,  #Same hight labels
     xlab = "", #Text of axis x
     ylab = "", #Text of axis y
     main = ""  #Text of drawing
    )
rect.hclust(groups, k=4)
```



hclust (*, "ward.D")

And the dendrogram with 10 aggrupations:

```
plot(groups,
     cex =0.25, #Size of labels
     hang= -1,  #Same hight labels
     xlab = "", #Text of axis x
     ylab = "", #Text of axis y
     main = ""  #Text of drawing
    )
rect.hclust(groups, k=10)
```



hclust (*, "ward.D")

2.2 Most frequent (and infrequent) features

The `quanteda` function `topfeatures()` provides the 10 most frequent (or infrequent) features (tokens) from a tfm. Let's apply it for our Quixote chapters.

```
topfeatures(dfm_capsQ)
```

```
    ,   que     y    de    la     a    en     .    el     -
39698 20416 17987 17953 10227  9718  8115  8089  8079  6918
```

Notice that the most common features include punctuation marks and *stop words*. To remove all this we can do:

```
#Without puntuation marks
dfm_capsQ_1  <- dfm(tokens(corpus_capsQ,
                           remove_punct = TRUE
                            #Default values:
                            # remove_punct = FALSE,
                            # remove_symbols = FALSE,
                            # remove_numbers = FALSE,
                            # remove_url = FALSE,
                            # remove_separators = TRUE,
                            # split_hyphens = FALSE
                          ),
                    #Default values:
                    # tolower = TRUE           #Convert to lowercase
                    # remove_padding  = FALSE  #Does padding (fill up blanks)
                  )
#Without stop words
```

```
dfm_capsQ_2 <- dfm_remove(dfm_capsQ_1, stopwords("es"))
topfeatures(dfm_capsQ_2)
```

```
     don   quijote    sancho        si      dijo       tan respondió       así
    2627      2155      2143      1938      1804      1220      1063      1059
     ser     señor
    1055      1054
```

The less frequent features are:

```
topfeatures(dfm_capsQ_2,
            decreasing = FALSE #By default it is TRUE
          )
```

```
 quebrantos     sábados    lantejas    palomino   consumían   concluían
          1           1           1           1           1           1
    velarte   pantuflos entresemana     vellorí
          1           1           1           1
```

2.3 Using docvars

We can take advantage of the corpus by filtering using docvars. For instance, if we are interested in comparing the most frequent *features* in the first part of Don Quixote and in the second part, we can do this:

```
corpus_part1 <- corpus_subset(corpus_capsQ,
                              Chapter < 53 #I keep chaps from 1 to 52
                             )
corpus_part2 <- corpus_subset(corpus_capsQ,
                              Chapter > 52 #I keep chaps from 53 to end
                             )

dfm_part1_noPunct  <- dfm(tokens(corpus_part1, remove_punct = TRUE))
dfm_part2_noPunct  <- dfm(tokens(corpus_part2, remove_punct = TRUE))

dfm_part1_noPunct_noSW <- dfm_remove(dfm_part1_noPunct, stopwords("es"))
dfm_part2_noPunct_noSW <- dfm_remove(dfm_part2_noPunct, stopwords("es"))

#Most frequent feat
topfeatures(dfm_part1_noPunct_noSW)
```

```
    don      si quijote    dijo     tan  sancho    bien     así     ser    pues
   1060     933     831     821     732     654     547     545     496     452
```

```
topfeatures(dfm_part2_noPunct_noSW)
```

```
     don    sancho   quijote        si      dijo     señor respondió       ser
    1567      1489      1324      1005       983       655       631       559
  merced       así
     525       514
```

```
#Less frequent feat
topfeatures(dfm_part1_noPunct_noSW, decreasing = FALSE)
```

```
   carnero   salpicón quebrantos    sábados   lantejas   palomino   domingos
         1          1          1          1          1          1          1
 consumían  concluían    velarte
         1          1          1
```

```
topfeatures(dfm_part2_noPunct_noSW, decreasing = FALSE)
```

```
    renovarle encargándolas     regalarle confortativas    apropiadas
            1             1             1             1             1
   parecerles     visitarle     acordaron   visitáronle       almilla
            1             1             1             1             1
```

3 Sentiment analysis

3.1 Analisys of English texts

From this Quanteda tutorial

You can detect occurrences of words in specific contexts by selectively applying a dictionary. In this example, we apply a sentiment dictionary to segments of news articles that mentions the (British) government.

In the package `quanteda.corpora` there is a corpus containing 6,000 Guardian news articles from 2012 to 2016.

```
require(quanteda)
require(quanteda.corpora)

corp_news <- download("data_corpus_guardian") #10.8MB disc, 29.2MB RAM
data_dictionary_LSD2015
```

```
Dictionary object with 4 key entries.
- [negative]:
  - a lie, abandon*, abas*, abattoir*, abdicat*, aberra*, abhor*, abject*, abnormal*, abolish*, abominal
- [positive]:
  - ability*, abound*, absolv*, absorbent*, absorption*, abundanc*, abundant*, acced*, accentuat*, accep
- [neg_positive]:
  - best not, better not, no damag*, no no, not ability*, not able, not abound*, not absolv*, not absorb
- [neg_negative]:
  - not a lie, not abandon*, not abas*, not abattoir*, not abdicat*, not aberra*, not abhor*, not abject
```

```
lengths(data_dictionary_LSD2015)#Useful for dictionaries
```

```
    negative     positive neg_positive neg_negative
        2858         1709         1721         2860
```

Tokenize texts and select tokens surrounding keywords related to the government using `tokens_keep()`:

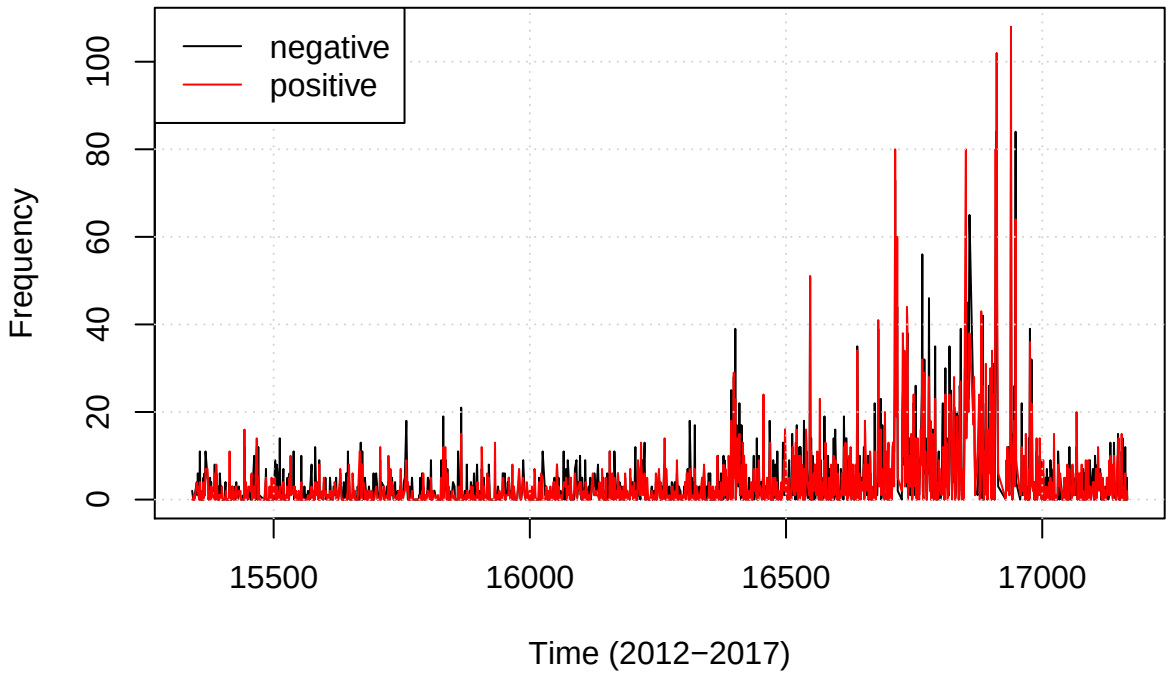
```r
# tokenize corpus
toks_news <- tokens(corp_news, remove_punct = TRUE)

# get relevant keywords and phrases
gov <- c("government", "cabinet", "prime minister")

# only keep tokens specified in the list and their context of ±10 tokens
# note: use phrase() to correctly score multi-word expressions
toks_gov <- tokens_keep(toks_news, pattern = phrase(gov), window = 10)
```

Apply the Lexicoder Sentiment Dictionary to the selected contexts using `tokens_lookup()`:

```r
# select only the "negative" and "positive" categories
data_dictionary_LSD2015_pos_neg <- data_dictionary_LSD2015[1:2]

toks_gov_lsd <- tokens_lookup(toks_gov,
                              dictionary = data_dictionary_LSD2015_pos_neg)

# create a document document-feature matrix and group it by date
dfmat_gov_lsd <- dfm(toks_gov_lsd) %>%
  dfm_group(groups = date)
```

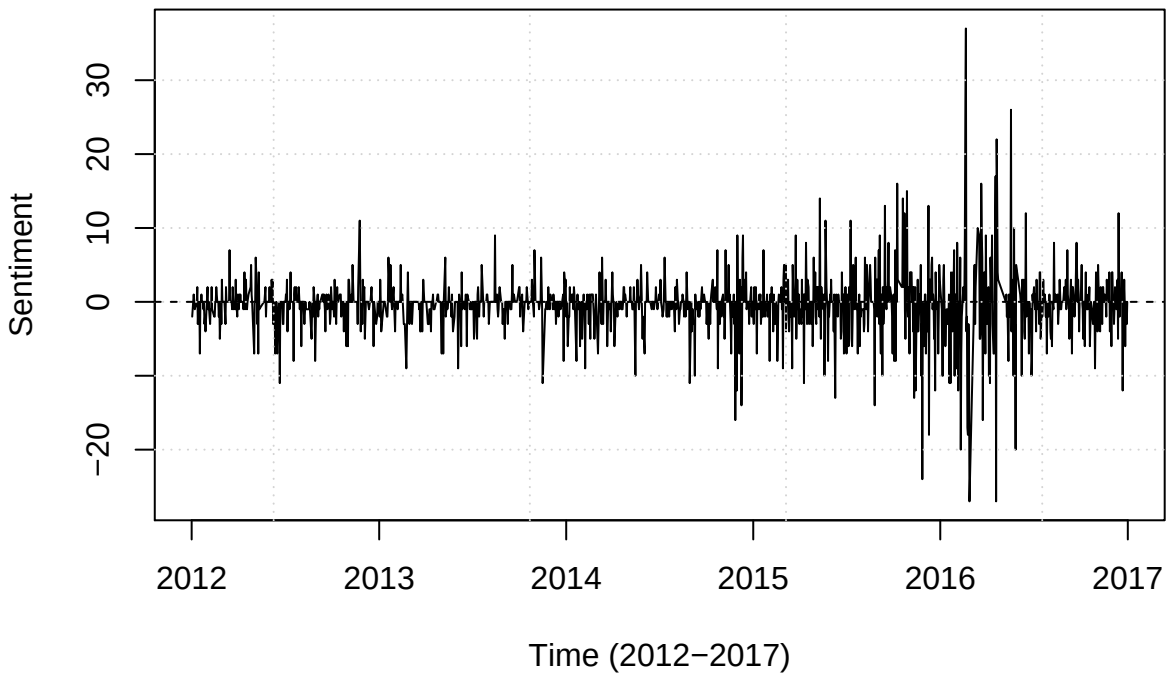
Make a nice plot:

```r
#matplot plots columns of matrices
matplot(dfmat_gov_lsd$date,
        dfmat_gov_lsd,
        type = "l",
        lty = 1,
        col = 1:2,
        xlab= "Time (2012-2017)",
        ylab = "Frequency")
grid()
legend("topleft", col = 1:2, legend = colnames(dfmat_gov_lsd), lty = 1, bg = "white")
```



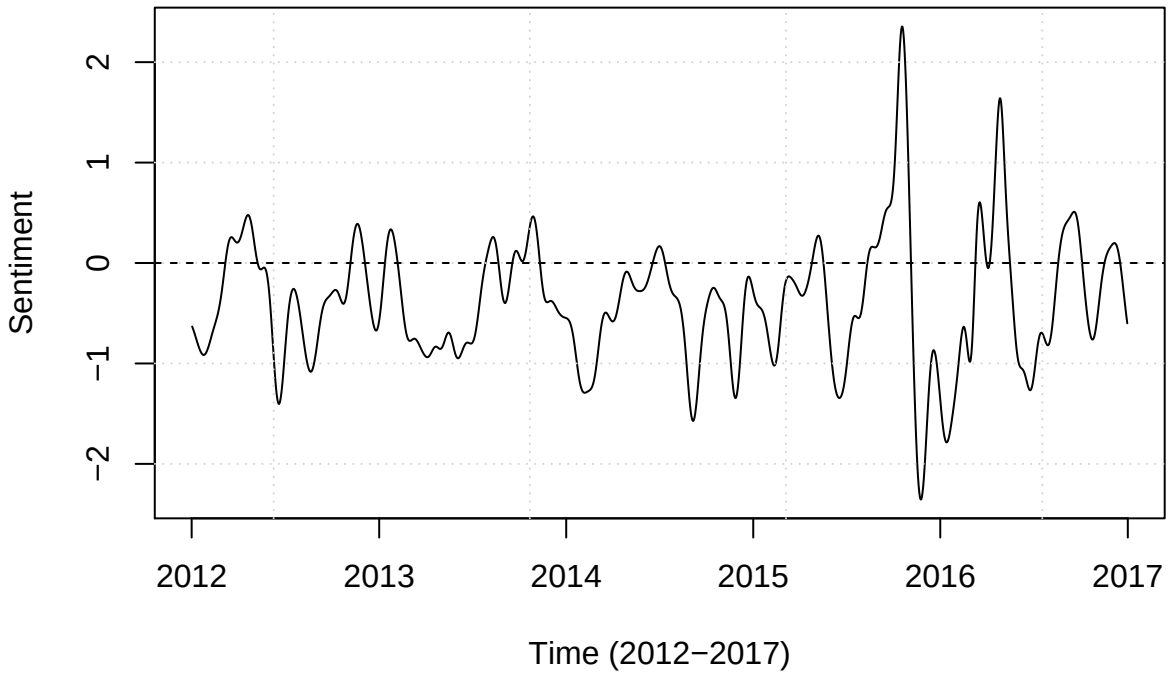
You can compute daily sentiment scores by taking the difference between the frequency of positive and negative words.

```
plot(dfmat_gov_lsd$date,
     dfmat_gov_lsd[,"positive"] - dfmat_gov_lsd[,"negative"],
     type = "l",
     ylab = "Sentiment",
     xlab = "Time (2012-2017)")
grid()
abline(h = 0, lty = 2)
```



You can apply kernel smoothing to show the trend more clearly:

```
dat_smooth <- ksmooth(x = dfmat_gov_lsd$date,
                      y = dfmat_gov_lsd[,"positive"] - dfmat_gov_lsd[,"negative"],
                      kernel = "normal",
                      bandwidth = 30)
plot(dat_smooth$x,
     dat_smooth$y,
     type = "l",
     ylab = "Sentiment",
     xlab = "Time (2012-2017)")
grid()
abline(h = 0, lty = 2)
```



3.2 Analysis of Spanish texts

3.2.1 Valence of texts in Spanish

Download the dictionary of sentiments for Spanish from here (file sentim_es.rds). This dictionary has *statistical descriptions* for valence (*valencia*), arousal (*excitación*), and concreteness (*concreción*), as well as five emotions: happiness (*felicidad*), anger (*ira*), sadness (*tristeza*), fearness (*miedo*) and disgust (*aversión*).

I have crated this dictionary from the data in the paper *Hinojosa, J. A. et al. (2016). Affective norms of 875 Spanish words for five discrete emotional categories and two emotional dimensions.*

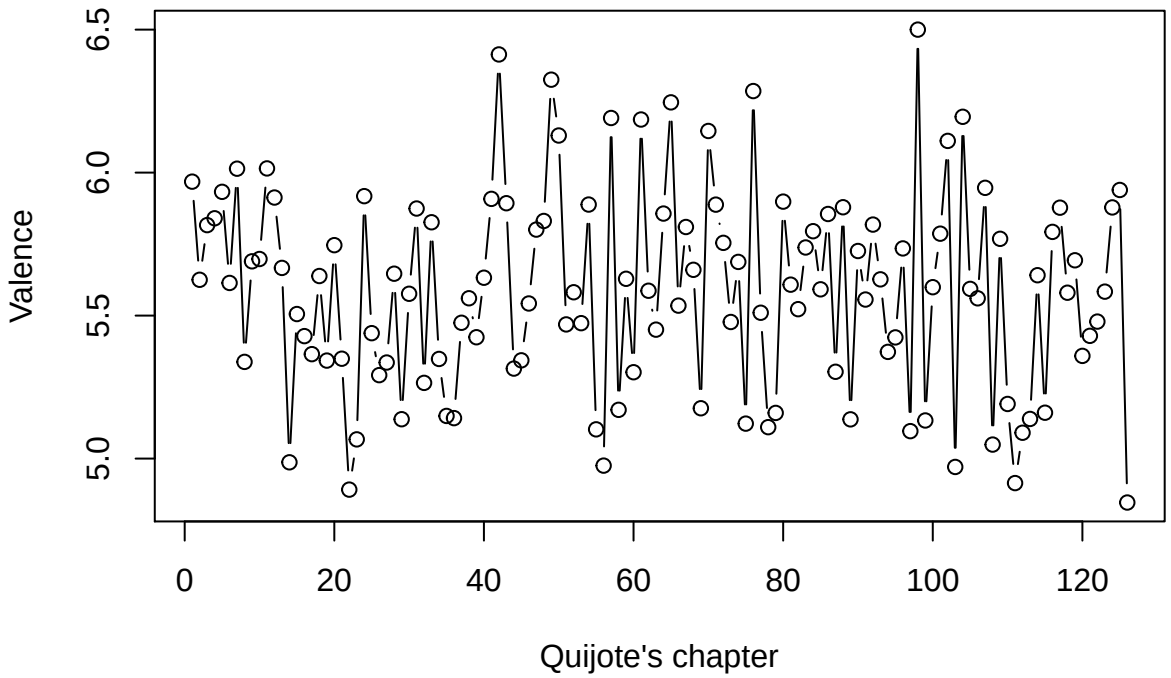
Lets's use it to analize the chapaters of El Quijote (file Qcaps.rds) from here).

```
data_dictionary_ANSW <- readRDS("sentim_es.rds") #ANSW = Affective Norms for Spanish Words

#Load the chapter texts and create the corpus
chaps <- readRDS(file="Qcaps.rds") #It is a list of strings
```

```
texts_chaps <- unlist(chaps)
names(texts_chaps) <- paste("Chap.", 1:length(texts_chaps)) #Assing a name to each string
library(quanteda)
corpus_chapsQ <- corpus(texts_chaps)
docvars(corpus_chapsQ, field="Chapter") <- 1:length(texts_chaps) #docvar with the number of the chapter

#Computes the valence of chapters
library(quanteda.sentiment) #remotes::install_github("quanteda/quanteda.sentiment")
vals <- textstat_valence(corpus_chapsQ,
                         dictionary = data_dictionary_ANSW["valence"])
```

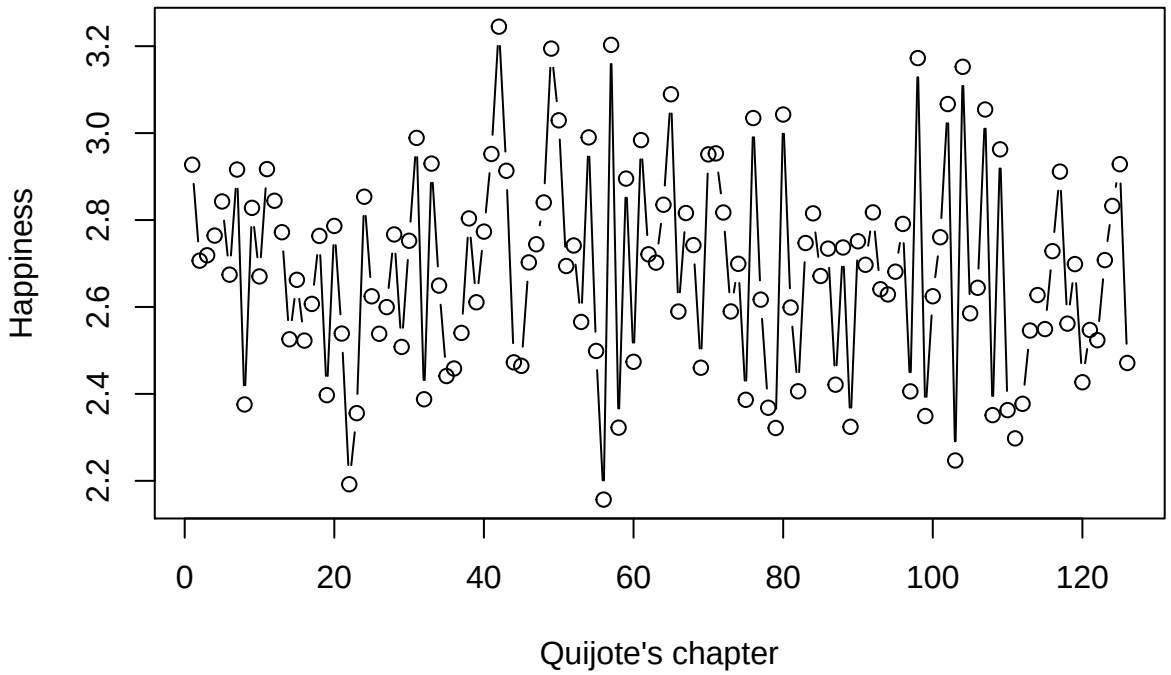
A simple graph. Pay attention:, `vals` is a `data.frame` in which column `doc_id` is a factor (the id of the string).

```
plot(x=1:nrow(vals),
     y=vals$sentiment,
     type="b", #b = both: dots and lines
     xlab="Quijote's chapter",
     ylab="Valence")
```



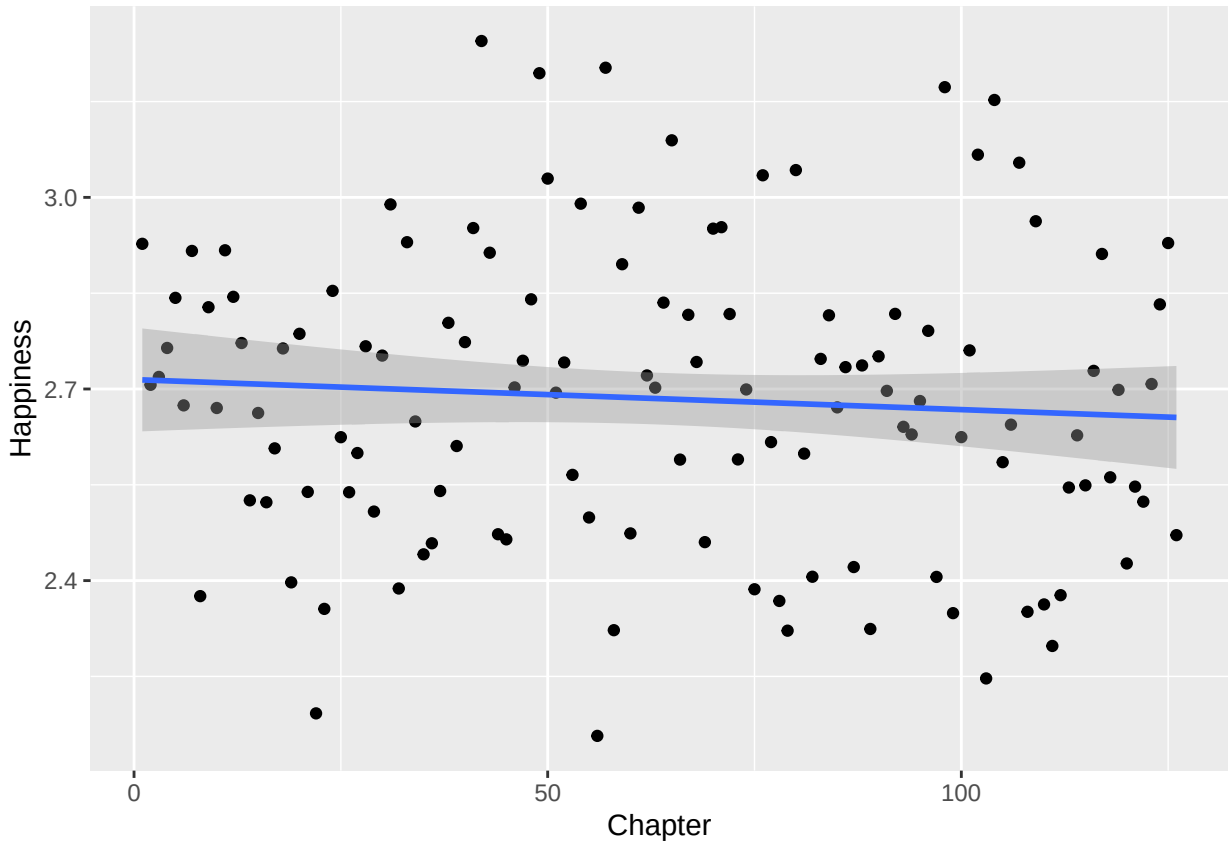
I can show the evolution of the feeling of happiness throughout the chapters:

```
#Compute the "happiness" of chapters
vals <- textstat_valence(corpus_chapsQ,
                         dictionary = data_dictionary_ANSW["happiness"])
plot(x=1:nrow(vals),
     y=vals$sentiment,
     type="b", #b = both: dots and lines
     xlab="Quijote's chapter",
     ylab="Happiness")
```



In order to show the trend I can use `ggplot2` package and the `geom_smooth()` layer.

```
library(ggplot2)
ggplot(vals, aes(1:nrow(vals), sentiment)) +
  xlab("Chapter") + ylab("Happiness") +
  geom_point() +
  geom_smooth(method = "glm") #The trend. It can be "lm" (linear smooth), "glm", "gam", etc.
```



I can also see the usage of the dictionary words in the corpus

```
#Words associated with "happiness" in the chapters
head(kwic(tokens(corpus_capsQ), data_dictionary_ANSW["happiness"]))
```

docname	from	to	pre	keyword	post	pattern
Chap. 1	94	94	las tres partes de su	hacienda	. El resto della concluían	happiness
Chap. 1	131	131	su vellorí de lo más	fino	. Tenía en su casa	happiness
Chap. 1	189	189	era de complexión recia ,	seco	de carnes , enjuto de	happiness
Chap. 1	239	239	conjeturas verosímiles , se deja	entender	que se llamaba Quejana .	happiness
Chap. 1	251	251	esto importa poco a nuestro	cuento	; basta que en la	happiness
Chap. 1	295	295	año , se daba a	leer	libros de caballerías , con	happiness

Notice that the words that `kwic` focuses on are the words in the dictionary. For example, you can see that `hacienda` is in the dictionary with this:

```
sum(unlist(as.list(data_dictionary_ANSW["happiness"])) == "hacienda") #Returns 1
```

```
[1] 1
```

3.2.2 Polarity of texts in Spanish

In order to compute polarity we will use the dictionary from here (file polar_es.rds). This dictionary has polarities from a Spanish lexicon (1555 positive words and 2720 negative words).

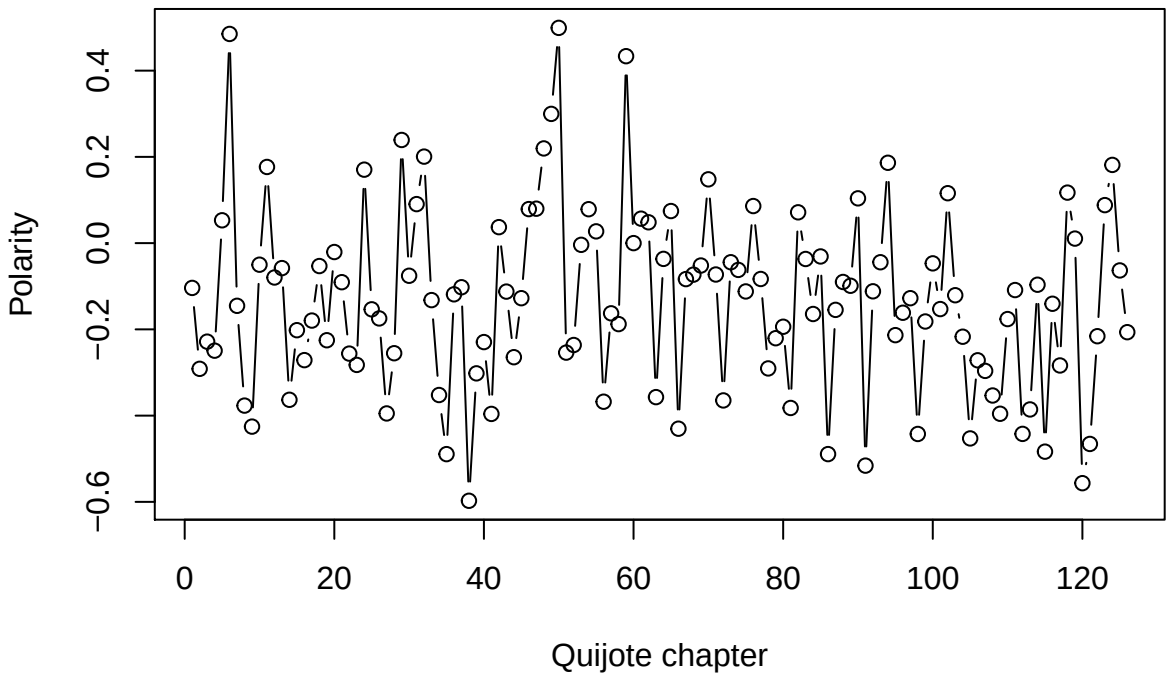
I have created this dictionary from the data in the paper *Chen, Y., & Skiena, S. (2014). Building Sentiment Lexicons for All Major Languages. In ACL (2) (pp. 383-389).* published in Kaggle (here).

To compute the polarity of Quijote chapters:

```
data_dictionary_AML_es <- readRDS("polar_es.rds") #AML = All Major Languages
#Computes polarity of chapters
library(quanteda.sentiment)
vals <- textstat_polarity(corpus_chapsQ,
                          dictionary = data_dictionary_AML_es)
```

You can plot the evolution of **polarity** throughout the chapters:

```
plot(x=1:nrow(vals),
     y=vals$sentiment,
     type="b", #b = both: dots and lines
     xlab="Quijote chapter",
     ylab="Polarity")
```



3.2.2.1 Polarity functions

By default, the `textstat_polarity()` function uses the `sent_logit()` function to calculate the polarity. You can see the details of this function in the documentation for `textstat_polarity()` (click the link on "sentiment-functions").

Use other types of functions, such as the built-in `sent_abspropdiff()` and `sent_relpropdiff()` or your own that use the `pos` and `neg` values.