Analizing Don Quixote and some other things

NLP master course 2021-2022

Mariano Rico (mariano.rico@upm.es)

Document created on 2022-01-09

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).

```r
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")
```

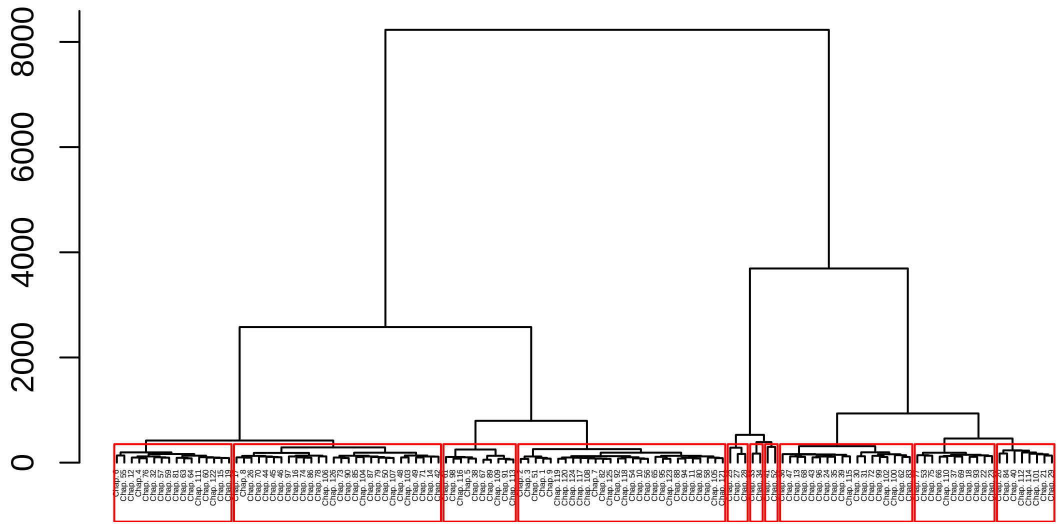
2.1 Dendrograms

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)
```

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
```

```r
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:

```r
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:

```r
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
```

```r
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
```

```r
#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 
```

```r
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 Document classification

We will use the dataset `data_corpus_LMRD` (from here (file data_corpus_LMRD.rds), which contains 50,000 movie reviews as texts and evaluations in the range [1,10] and as `pos` (positive) or `neg` (negative). This R object is a quanteda corpus with reviews as documents and 4 docvars (docnumber, rating, set and polarity). It is important the `set` docvar with values `train` and `test`.

The objective is to create a classification model. We will use the reviews labeled as `train` to train a model, while reviews labeled as `test` to test the model and compute the confusion matrix. In this example, the predicted values will be the polarity (`pos` or `neg`) of the review.

3.1 Naive Bayes

The simplest predictive model is *Naive Bayes*, that can be used with the function `textmodel_nb()` (in package `quanteda.textmodels`). Notice that this function has an argument `distribution` that can be `distribution = "multinomial"` (default value) or `distribution = "Bernoulli"`.

We can be interested in knowing the values of *precision*, *recall* and *accuracy* for both alternatives.

```r
library(quanteda) #Required to read a corpus object
data_corpus_LMRD <- readRDS("data_corpus_LMRD.rds")
dfmat <- dfm(tokens(data_corpus_LMRD)) #50.000 docs x 149.653 feats
#Only uses 106MB RAM
dfmat_train <- dfm_subset(dfmat, set == "train")
dfmat_test <- dfm_subset(dfmat, set == "test")

library(quanteda.textmodels) #For textmodel_nb()
library(caret) #For confusionMatrix()
nbPredictions <- function(dist){ #dist = "multinomial" or "Bernoulli"
  #Compute a nb (Naive Bayes) model
  multi <- textmodel_nb(dfmat_train,
                        dfmat_train$polarity,
                        distribution = dist)
  #Predictions with the model
  pred <- predict(multi, #the computed model
                  newdata = dfmat_test)
```

```r
  #Compute the confusion matriz for our prediction
  confM <- confusionMatrix(pred, docvars(dfmat_test)$polarity)

  #Accuracy is the number of labels (strings) that match...
  my_acc_coincidences <- sum(as.character(pred) == as.character(docvars(dfmat_test)$polarity))
  #...divided by the total number of labels
  my_acc_total <- length(as.character(pred))
  my_acc <- my_acc_coincidences/my_acc_total
  my_acc #Sale 0.82876. Con "multinomial" era 0.81304

  #Precision
  precision <- confM$byClass['Pos Pred Value']
  precision #Sale 0.7951591 (antes  0.878327)
  #Recall
  recall <- confM$byClass['Sensitivity']
  recall #Sale 0.88568 (antes 0.8953488)
  list(acc = my_acc, p = precision, r = recall)
}
nbPredictions("multinomial")
```

```
$acc
[1] 0.81304

$p
Pos Pred Value
     0.7763418

$r
Sensitivity
    0.87944
```

```r
nbPredictions("Bernoulli")
```

```
$acc
[1] 0.82876

$p
Pos Pred Value
     0.7951591

$r
Sensitivity
    0.88568
```

We can see that `Bernoulli` provides slightly better values than `multinomial`.

If i use tf-idf, will get better results?

```r
#Like before but using tf-idf
dfmat <- dfm_tfidf(dfmat, #Over the previous dfmat, compute tf-idf
                   scheme_tf = "prop" #By default it is "count". The good one for TF-IDF is "prop"
                  )
dfmat_train <- dfm_subset(dfmat, set == "train")
```

```r
dfmat_test <- dfm_subset(dfmat, set == "test")
nbPredictions("multinomial")
```

```
$acc
[1] 0.85236

$p
Pos Pred Value
     0.8316392

$r
Sensitivity
     0.8836
```

```r
nbPredictions("Bernoulli")
```

```
$acc
[1] 0.82876

$p
Pos Pred Value
     0.7951591

$r
Sensitivity
    0.88568
```

We get the same results. Probably these methods compute the TF-IDF internally.

3.2 SVM Model

Now we will use the function `textmodel_svm()` to create a SVM model. Will we get better results (better *precision*, *recall* or *accuracy*) than the *Naive Bayes* model?

```r
#Compute a SVM model
multi <- textmodel_svm(dfmat_train,
                       dfmat_train$polarity,
                       weight = "uniform") #Default value. Other options: "docfreq", "termfreq".
```

Quickly we get this error:

`Cholmod error 'problem too large' at file ../Core/cholmod_dense.c, line 102.`

It seems that this a too big problem for this package `:-(`

We can try with a smaller training dataset. the `dfm_sample ()` function allows you to take a sample of the size you want.

```r
set.seed(123) #Reproducible results

svmPredictions <- function(x,       #Sample size
                           weight){ #weight can be "uniform", "docfreq" or "termfreq".
```

```
  #Instead of using all the documents marked as train, I take only x documents
  dfmat_train <- dfm_sample(dfm_subset(dfmat, set == "train"),
                             x #Sample size
                            )
  dfmat_test <- dfm_subset(dfmat, set == "test")
  multi <- textmodel_svm(dfmat_train,
                         dfmat_train$polarity,
                         weight = weight)
  pred <- predict(multi,
                  newdata = dfmat_test)
  confM <- confusionMatrix(pred, docvars(dfmat_test)$polarity)

  my_acc_coincidences <- sum(as.character(pred) == as.character(docvars(dfmat_test)$polarity))
  my_acc_total <- length(as.character(pred))
  my_acc <- my_acc_coincidences/my_acc_total

  precision <- confM$byClass['Pos Pred Value']

  recall <- confM$byClass['Sensitivity']

  list(acc = my_acc, p = precision, r = recall)
}

svmPredictions(10000, "uniform")
```

```
Warning: 84304 features in newdata not used in prediction.

$acc
[1] 0.85312

$p
Pos Pred Value
     0.8585703

$r
Sensitivity
    0.84552
```

```
svmPredictions(10000, "docfreq")
```

```
Warning: 84012 features in newdata not used in prediction.

$acc
[1] 0.83424

$p
Pos Pred Value
     0.8901027

$r
Sensitivity
    0.76264
```

```r
#svmPredictions(10000, "termfreq") #Produces an error
```

Does anyone dare to make a graph showing on the x-axis the number of samples taken, and in the y-axis the value of accuracy, precision and recall?.