Text mining in practice - Bag of Words - stopwords

Digital Humanities DSC 105 Spring 2023

Marcus Birkenkrahe

February 23, 2023

README

- This lecture closely follows the 3rd part of the DataCamp lesson "Jumping into Text Minin with Bag-of-Words" by Ted Kwartler, part of his course on "Text Mining with Bag-of-Words in R".
- Download and open the practice file `5_stopwords_practice.org` from GitHub to code along.

Getting the coffee data

Run this in case you had to interrupt the previous session and don't have the data in your R session:

```
library(tm)
coffee_df <- read.csv("../data/coffee.csv") # dataframe
coffee_vec <- coffee_df$text # vector
coffee_src <- VectorSource(coffee_vec) # source
coffee_corpus <- VCorpus(coffee_src)
```

All about stop words

- Load the `tm` package and look for the `stopwords` function:

  ```
  library(tm)
  ## is stopwords any of the functions in tm?
  ```

```
f_tm <- ls('package:tm') # store all function names in f_tm
any(f_tm=="stopwords") # check every function against "stopwords"
```

```
[1] TRUE
```

- The function `any` is very useful: it checks if any of its arguments are true:

```
any(c(T,F,F,F,F,F)==TRUE)
any(c(F,F,F)==TRUE)
any("Joe" %in% c("Jim","Joe","Jane")) # is Joe in the team?
any("Josh" %in% c("Jim","Joe","Jane")) # is Josh in the team?
```

```
[1] TRUE
[1] FALSE
[1] TRUE
[1] FALSE
```

- Check out the `stopwords` in English ("en" or "english"), Spanish ("es"), German ("de" or "german").

```
stopwords("en")
```

```
  [1] "i"          "me"         "my"         "myself"     "we"
  [6] "our"        "ours"       "ourselves"  "you"        "your"
 [11] "yours"      "yourself"   "yourselves" "he"         "him"
 [16] "his"        "himself"    "she"        "her"        "hers"
 [21] "herself"    "it"         "its"        "itself"     "they"
 [26] "them"       "their"      "theirs"     "themselves" "what"
 [31] "which"      "who"        "whom"       "this"       "that"
 [36] "these"      "those"      "am"         "is"         "are"
 [41] "was"        "were"       "be"         "been"       "being"
 [46] "have"       "has"        "had"        "having"     "do"
 [51] "does"       "did"        "doing"      "would"      "should"
 [56] "could"      "ought"      "i'm"        "you're"     "he's"
 [61] "she's"      "it's"       "we're"      "they're"    "i've"
 [66] "you've"     "we've"      "they've"    "i'd"        "you'd"
 [71] "he'd"       "she'd"      "we'd"       "they'd"     "i'll"
 [76] "you'll"     "he'll"      "she'll"     "we'll"      "they'll"
```

```
 [81] "isn't"      "aren't"     "wasn't"     "weren't"    "hasn't"
 [86] "haven't"    "hadn't"     "doesn't"    "don't"      "didn't"
 [91] "won't"      "wouldn't"   "shan't"     "shouldn't"  "can't"
 [96] "cannot"     "couldn't"   "mustn't"    "let's"      "that's"
[101] "who's"      "what's"     "here's"     "there's"    "when's"
[106] "where's"    "why's"      "how's"      "a"          "an"
[111] "the"        "and"        "but"        "if"         "or"
[116] "because"    "as"         "until"      "while"      "of"
[121] "at"         "by"         "for"        "with"       "about"
[126] "against"    "between"    "into"       "through"    "during"
[131] "before"     "after"      "above"      "below"      "to"
[136] "from"       "up"         "down"       "in"         "out"
[141] "on"         "off"        "over"       "under"      "again"
[146] "further"    "then"       "once"       "here"       "there"
[151] "when"       "where"      "why"        "how"        "all"
[156] "any"        "both"       "each"       "few"        "more"
[161] "most"       "other"      "some"       "such"       "no"
[166] "nor"        "not"        "only"       "own"        "same"
[171] "so"         "than"       "too"        "very"
```

- Check yourself if the word "should" is in `stopwords("en")`:

```
any(stopwords("en")=="should")
```

```
[1] TRUE
```

- Add two stop words to `stopwords("en")` and check that they were added:

  1. append "word1" and "word2" to `stopwords("en")` using `c()`
  2. store the result in `all_stops`
  3. display the first two entries of `all_stops`

```
all_stops <- c("word1", "word2", stopwords("en"))
head(all_stops,2)
```

```
[1] "word1" "word2"
```

- To remove words, you can use `tm::removeWords`. It takes two *arguments*: the text object to which it is applied, and the list of words to remove.
- List the arguments of `removeWords`.

```
args(removeWords)
```

```
function (x, words)
NULL
```

- You see that there are two arguments: `x` is the input dataset, and `words` are the words to be removed as `character` strings.

Exercise with `stopwords`

- Remove all `stopwords` from sample `text`, add two words to the standard `stopwords` dictionary, and remove them from `text`, too.
- Define sample `text` vector.

```
text <-
  "<b>She</b> woke up at       6 A.M. It\'s so
   early!  She was only 10% awake and began drinking
   coffee in front of her computer."
text
```

```
[1] "<b>She</b> woke up at       6 A.M. It's so\n   early!  She was only 10% awake
```

- Remove "en" stopwords from `text` with `removeWord`.

```
text
removeWords(text, stopwords("en"))
```

```
[1] "<b>She</b> woke up at       6 A.M. It's so\n   early!  She was only 10% awake
[1] "<b>She</b> woke         6 A.M. It's \n   early!  She   10% awake  began drink
```

- How many words were removed? Use `nchar` to check.
- Remove "She" from `text`:

- Add "coffee" and "bean" to the standard stop words and assign the result to `new_stops`. Check that they are in `new_stops`!

```
new_stops <- c("coffee", "bean", stopwords("en"))
head(new_stops,2)
```

```
[1] "coffee" "bean"
```

- Wait a moment! What if these words were already in `stopwords`?

  1. save `stopwords("en")` as `old_stops`
  2. check if any elements of `old_stops` are "coffee" or "bean"
  3. check if any elements of `new_stops` are "coffee" or "bean"

```
old_stops <- stopwords("en") # store old stopwords in old_stops
any(old_stops=="coffee"|old_stops=="bean")
any(new_stops=="coffee"|new_stops=="bean")
```

```
[1] FALSE
[1] TRUE
```

- Remove the customized stopwords, `new_stops`, from `text`:

```
text
removeWords(text, new_stops)
```

```
[1] "<b>She</b> woke up at       6 A.M. It's so\n   early!  She was only 10% awake
[1] "<b>She</b> woke         6 A.M. It's \n   early!  She   10% awake  began drink
```

Interlude: finding a string in a dataset

- To find a tweet in `coffee_vec` that contains both words, we need a few more tricks: index vectors with `which` and pattern search with `grepl`.
- `which` runs its `logical` argument a vector and returns the indices that satisfy the logical argument:

```
foo <- c(10,20,30,40,50)  # sample vector
which (foo == 20)  # which elements of x are equal 2?
which (foo >= 30)  # which elements of x are greater or equal to 3?
```

```
[1] 2
[1] 3 4 5
```

- The same thing works with `character` vectors:

```
bar <- c("High", "Noon", "in", "Batesville")
which (bar == "High")  # elements of bar equal "High"
which (bar == "Batesville" | # elements of bar either
       bar == "in")          # equal "Batesville" or equal "in"
```

```
[1] 1
[1] 3 4
```

- It also works with `stopwords`: e.g. is "cannot" in the `stopwords` vector? And which index of the `stopwords` vector is it?

```
str(stopwords()) # structure
idx <- which(stopwords("en") == "cannot") # index vector
stopwords("en")[idx] # extract the element no. idx
```

```
 chr [1:174] "i" "me" "my" "myself" "we" "our" "ours" "ourselves" "you" ...
[1] "cannot"
```

- `grepl` checks if its `pattern` is contained in a dataset `x`. It returns a `logical` vector, a matrh or not for each element of `x`:

```
args(grepl)
```

```
function (pattern, x, ignore.case = FALSE, perl = FALSE, fixed = FALSE,
    useBytes = FALSE)
NULL
```

- For example: check if any coffee tweets contain the word "Ramadan"

```
any(grepl(pattern="Ramadan",x=coffee_vec))
```

```
Error in is.factor(x) : object 'coffee_vec' not found
```

- Combine `grepl` and `which` to extract the corresponding index:

```
which(grepl(pattern="Ramadan",x=coffee_vec))

Error in is.factor(x) : object 'coffee_vec' not found
```

- Then print the corresponding tweets:

```
idx <- which(grepl(pattern="Ramadan",x=coffee_vec))
coffee_vec[idx]

Error in is.factor(x) : object 'coffee_vec' not found
Error: object 'coffee_vec' not found
```

Finding certain tweets in `coffee_vec`

- Now, to find the tweets in `coffee_vec` that contain "coffee" AND "beans":
  1. create an index vector of tweets that contain "beans"
  2. store these tweets in `bean`
  3. create an index vector of `bean` tweets that contain "coffee"
  4. store these tweets in `coffee`

```
idx_bean <- which(grepl("bean",coffee_vec))
bean <- coffee_vec[idx_bean] # all tweets with "bean"
idx_coffee_bean <- which(grepl("coffee",bean))
coffee_bean <- bean[idx_coffee_bean]
coffee_bean

Error in is.factor(x) : object 'coffee_vec' not found
Error: object 'coffee_vec' not found
Error in is.factor(x) : object 'bean' not found
Error: object 'bean' not found
Error: object 'coffee_bean' not found
```

- Now re-run the code above to remove "bean" and "coffee" from the selection `coffee_bean`:

```
removeWords(coffee_bean, new_stops)

Error in removeWords(coffee_bean, new_stops) :
  object 'coffee_bean' not found
```

Word stemming on a sentence

- If you call `stemDocument` on a sentence it fails. Try it with the sample text:

```
sentence <- "In a complicated haste,
  Tom rushed to fix a new complication,
  too complicatedly."
sentence
```

```
[1] "In a complicated haste,\n  Tom rushed to fix a new complication,\n  too compl
```

- Alas, I wrote this over several lines and it contains newline characters `\n` - white space - do you know how to remove it?

```
sentence <- stripWhitespace(sentence)
sentence
```

```
[1] "In a complicated haste, Tom rushed to fix a new complication, too complicated
```

- Now run `stemDocument` on the `sentence`:

```
stemDocument(sentence)
```

```
[1] "In a complic haste, Tom rush to fix a new complication, too complicatedly."
```

- This happens because `stemDocument()` treats the whole sentence **as one word**: the document is a `character` vector of length 1:

```
is.vector(sentence)
length(sentence)
```

```
[1] TRUE
[1] 1
```

- To solve this problem

  1. remove the punctuation marks with `removePunctuation`
  2. split the `sentence` in individual words using `strsplit`
  3. re-apply `stemDocument` and `stemCompletion` with our dictionary

Interlude: Splitting strings with `strsplit`

- To split strings, `strsplit` is handy. The only problem is that it returns a `list` instead of a vector so we have to `unlist` the result
- It is helpful for a new function to check the `help` (if you run the code block below, a browser will open and you'll have to stop the process in Emacs with `C-g`):

```
help(base::strsplit)
```

- What did you learn? `x` is the target data set, and `split` is a vector used for splitting. Never mind about the other arguments!

```
args(strsplit)
```

```
function (x, split, fixed = FALSE, perl = FALSE, useBytes = FALSE)
NULL
```

- For example, split this sentence: "Split this sentence" using "" as the `split` argument:

```
foo <- "Split this sentence"
strsplit(foo," ")
```

```
[[1]]
[1] "Split"    "this"     "sentence"
```

- That didn't quite work. What's the correct `split` to get the words?

```
bar <- strsplit(s," ")
bar
```

```
Error in strsplit(s, " ") : object 's' not found
[1] "High"       "Noon"       "in"         "Batesville"
```

- Now, the result of the split is a `list` and needs to be un-listed:

```
class(bar)
bar |> unlist() |> class()
```

```
[1] "character"
[1] "character"
```

- Just for fun, can you turn the pipeline in the last code block into a nested statement?

```
class(unlist(bar))
```

```
[1] "character"
```

Stem and re-complete a sentence

- Now, we're ready to deliver on our earlier promise:
  1. remove the punctuation marks with `removePunctuation`
  2. split the `sentence` in individual words using `strsplit`
  3. re-apply `stemDocument` and `stemCompletion` with our dictionary
- Sample sentence and sample dictionary for stem re-completion:

```
sentence <- stripWhitespace("In a complicated haste,
                            Tom rushed to fix a new complication,
                            too complicatedly.")
sentence
comp_dict <- c("In","a","complicate","haste",
               "Tom","rush","to","fix","new","too")
comp_dict
```

```
[1] "In a complicated haste, Tom rushed to fix a new complication, too complicated
 [1] "In"         "a"          "complicate" "haste"      "Tom"
 [6] "rush"       "to"         "fix"        "new"        "too"
```

- Remove the punctuation marks in `sentence` using `removePunctuation()`, and assign the result to `foo`:

```
foo <- removePunctuation(sentence)
foo
```

```
[1] "In a complicated haste Tom rushed to fix a new complication too complicatedly
```

- Call `strsplit()` on `foo` with the split argument set equal to " ", and save the result to `bar`:

```
bar <- strsplit(x = foo,
                split = " ")
bar
```

```
[[1]]
 [1] "In"            "a"             "complicated"   "haste"
 [5] "Tom"           "rushed"        "to"            "fix"
 [9] "a"             "new"           "complication"  "too"
[13] "complicatedly"
```

- Finally, unlist `bar`, assign the result to `baz` and test that `baz` is a `character` vector:

```
bar |> unlist() -> baz
baz |> is.character()
baz |> is.vector()
```

```
[1] TRUE
[1] TRUE
```

- Exercise: can you do the three steps - `removePunctuation`, `strsplit` and `unlist` in one command starting with `sentence`?

```
unlist(strsplit(removePunctuation(sentence)," "))
```

```
 [1] "In"            "a"             "complicated"   "haste"
 [5] "Tom"           "rushed"        "to"            "fix"
 [9] "a"             "new"           "complication"  "too"
[13] "complicatedly"
```

- Back to the main course: use `stemDocument` on `baz` and assign the result to `stem_doc`:

```
stem_doc <- stemDocument(baz)
stem_doc
```

```
[1] "In"      "a"       "complic" "hast"    "Tom"     "rush"    "to"
[8] "fix"     "a"       "new"     "complic" "too"     "complic"
```

- Re-complete the stemmed document with `stemCompletion` using `comp_dict` as reference dictionary and save the result in `complete_doc`:

```
complete_doc <- stemCompletion(stem_doc,comp_dict)
complete_doc
```

```
          In            a      complic         hast          Tom         rush
        "In"          "a" "complicate"      "haste"        "Tom"       "rush"
          to          fix            a          new      complic          too
        "to"        "fix"          "a"        "new" "complicate"        "too"
     complic
"complicate"
```

- This is the expected result: `complete_doc` is a named `character` vector whose names are the word stems (only `complic` was stemmed), and whose values are the completed words.

```
str(complete_doc)
```

```
Named chr [1:13] "In" "a" "complicate" "haste" "Tom" "rush" "to" "fix" "a" ...
- attr(*, "names")= chr [1:13] "In" "a" "complic" "hast" ...
```

Apply preprocessing steps to a corpus

- Earlier, we met the function `tm_map` to apply cleaning functions to an entire corpus. Here, we use it to clean out stop words.
- Reload the coffee corpus if you don't have it anymore in your R session - check this:

```
any(ls()=="coffee_corpus")
any(search()=="package:qdap")
any(search()=="package:tm")
```

```
[1] FALSE
[1] TRUE
[1] TRUE
```

- Reload it in case and load the necessary libraries, then run the search again:

```
library(tm)
coffee_df <- read.csv("../data/coffee.csv") # dataframe
coffee_vec <- coffee_df$text # vector
coffee_src <- VectorSource(coffee_vec) # source
coffee_corpus <- VCorpus(coffee_src)
any(ls()=="coffee_corpus")
any(search()=="package:qdap")
any(search()=="package:tm")
```

```
Error in file(file, "rt") : cannot open the connection
In addition: Warning message:
In file(file, "rt") :
  cannot open file '../data/coffee.csv': No such file or directory
Error: object 'coffee_df' not found
Error in SimpleSource(length = length(x), content = x, class = "VectorSource") :
  object 'coffee_vec' not found
Error in stopifnot(inherits(x, "Source")) : object 'coffee_src' not found
[1] FALSE
[1] TRUE
[1] TRUE
```

- To apply a cleaning function to the corpus, call it on the corpus and add the function as an argument.
- Example: remove the numbers from tweet no. 2:

```
corpus <- tm_map(coffee_corpus,removeNumbers)  # remove numbers
content(coffee_corpus[[2]]) # original tweet
content(corpus[[2]])  # tweet with numbers removed
```

```
Error in tm_map(coffee_corpus, removeNumbers) :
  object 'coffee_corpus' not found
Error in content(coffee_corpus[[2]]) : object 'coffee_corpus' not found
Error in content(corpus[[2]]) : object 'corpus' not found
```

- To apply more than one cleaning function to a corpus, we create our own custom function, `clean_corpus`. Here is what it does:

1. tm's `removePunctuation()`.
2. Base R's `tolower()`.
3. Remove the word "coffee" with `tm::removeWords`
4. Remove all white space with `tm::stripWhitespace`

```
clean_corpus <- function(corpus) {
  corpus <- tm_map(corpus,
                   removePunctuation)
  corpus <- tm_map(corpus,
                   content_transformer(tolower))
  corpus <- tm_map(corpus,
                   removeWords,
                   words = c(stopwords("en"), "coffee"))
  corpus <- tm_map(corpus,
                   stripWhitespace)
  return(corpus)
}
```

- The function `clean_corpus` will now run all its content functions on any corpus argument - to test this:

1. run `clean_corpus` on `coffee_corpus` and save it as `clean_coffee`
2. print the cleaned 227th tweet using `[[` and `content`
3. Compare it to the original tweet from `coffee_corpus`.

```
clean_corp <- clean_corpus(coffee_corpus)
content(clean_corp[[999]]) # lower case, no punctuation, no stopwords,
                           # no "coffee"
content(coffee_corpus[[999]])
```

```
Error in tm_map(corpus, removePunctuation) :
  object 'coffee_corpus' not found
Error in content(clean_corp[[999]]) : object 'clean_corp' not found
Error in content(coffee_corpus[[999]]) : object 'coffee_corpus' not found
```