Text mining - Bag of Words

DSC205 Introduction to Advanced Data Science

Marcus Birkenkrahe

April 1, 2023

README

- Short introduction to text mining (TM) and bag-of-words
- Based on Kwartler, Text mining in practice with R (Wiley, 2019)
- Kwartler is also the author of the DataCamp course on TM with R

Quick taste of test mining with `qdap`

This is a quick example to demonstrate text mining using the `qdap` package that you need to install first:

- (Install and) load `qdap` and show all loaded packages. Run this code block twice.

```
library(qdap)
search()
```

```
Loading required package: qdapDictionaries
Loading required package: qdapRegex
Loading required package: qdapTools
Loading required package: RColorBrewer

Attaching package: 'qdap'

The following objects are masked from 'package:tm':
```

```
    as.DocumentTermMatrix, as.TermDocumentMatrix

The following object is masked from 'package:NLP':

    ngrams

The following objects are masked from 'package:base':

    Filter, proportions
 [1] ".GlobalEnv"               "package:qdap"
 [3] "package:RColorBrewer"     "package:qdapTools"
 [5] "package:qdapRegex"        "package:qdapDictionaries"
 [7] "package:tm"               "package:NLP"
 [9] "ESSR"                     "package:stats"
[11] "package:graphics"         "package:grDevices"
[13] "package:utils"            "package:datasets"
[15] "package:stringr"          "package:httr"
[17] "package:methods"          "Autoloads"
[19] "package:base"
```

- Store this text in a vector `text`:

  > "DataCamp is the first online learning platform that focuses on building the best learning experience specifically for Data Science. We have offices in New York, London, and Belgium, and to date, we trained over 11 million (aspiring) data scientists in over 150 countries. These data science enthusiasts completed more than 667 million exercises. You can take free beginner courses, or subscribe for $25/month to get access to all premium courses."

  In Emacs:

  1. Mark the start of the text with `C-SPC` (CTRL + SPACEBAR)
  2. Go down to the end of the text with `C-e` (CTRL + e)
  3. Copy the text with `M-w` (ALT + w)
  4. Paste the text wherever you want to with `C-y` (CTRL + y)

```
text <- "DataCamp is the first online learning platform that focuses on building t
```

- Print `text`.

```
text
```

```
[1] "DataCamp is the first online learning platform that focuses on building the k
```

- Check the data type of `text` with `class`, and print its `length` and the number of its characters with `nchar`:

```
class(text)
length(text)
nchar(text)
```

```
[1] "character"
[1] 1
[1] 446
```

- Find the 10 most frequent terms and store them in `term_count` using `qdap::freq_terms`:

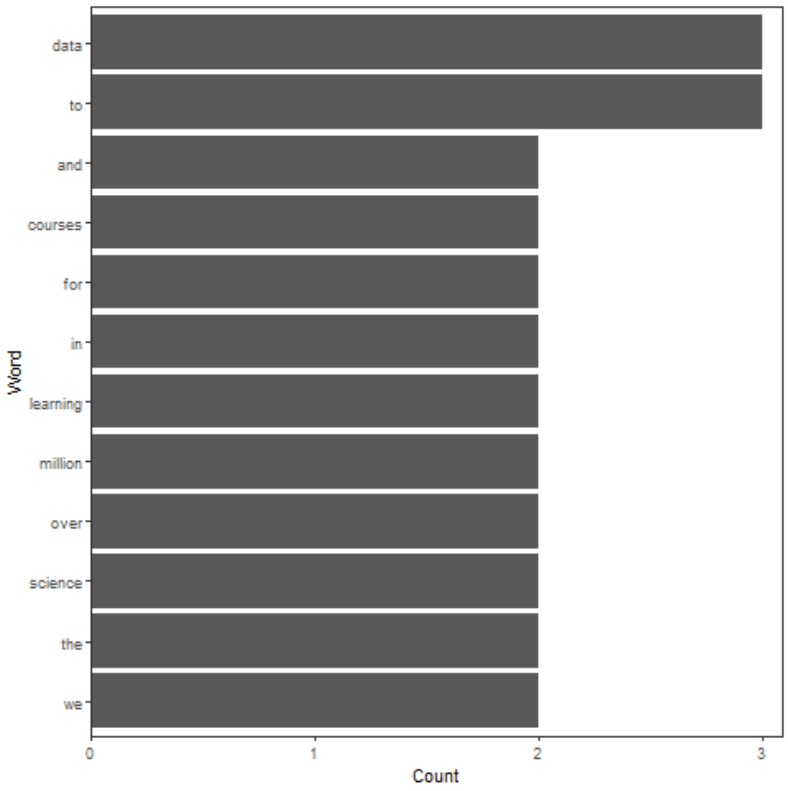
```
term_count <- freq_terms(text, 10)
term_count
```

```
   WORD     FREQ
1  data        3
2  to          3
3  and         2
4  courses     2
5  for         2
6  in          2
7  learning    2
8  million     2
9  over        2
10 science     2
11 the         2
12 we          2
```

- If you compare with what we said above, you can see that this table is a transposed document matrix (TDM) with one feature (word frequency `FREQ`).

- Plot the term count:

```
plot(term_count)
```



The tm text mining package

- The tm package for text mining comes with a *vignette* (Feinerer, 2022). Its date reveals that the paper is up to date.
- Load tm (twice) and check the loaded package list with search():

```
library(tm)
search()
```

```
 [1] ".GlobalEnv"                "package:qdap"
 [3] "package:RColorBrewer"      "package:qdapTools"
```

```
 [5] "package:qdapRegex"        "package:qdapDictionaries"
 [7] "package:tm"               "package:NLP"
 [9] "ESSR"                     "package:stats"
[11] "package:graphics"         "package:grDevices"
[13] "package:utils"            "package:datasets"
[15] "package:stringr"          "package:httr"
[17] "package:methods"          "Autoloads"
[19] "package:base"
```

- There is no separate data package. Check which functions `tm` contains:

```
ls("package:tm")
```

```
 [1] "as.DocumentTermMatrix"   "as.TermDocumentMatrix"
 [3] "as.VCorpus"              "Boost_tokenizer"
 [5] "content_transformer"     "Corpus"
 [7] "DataframeSource"         "DirSource"
 [9] "Docs"                    "DocumentTermMatrix"
[11] "DublinCore"              "DublinCore<-"
[13] "eoi"                     "findAssocs"
[15] "findFreqTerms"           "findMostFreqTerms"
[17] "FunctionGenerator"       "getElem"
[19] "getMeta"                 "getReaders"
[21] "getSources"              "getTokenizers"
[23] "getTransformations"      "Heaps_plot"
[25] "inspect"                 "MC_tokenizer"
[27] "nDocs"                   "nTerms"
[29] "PCorpus"                 "pGetElem"
[31] "PlainTextDocument"       "read_dtm_Blei_et_al"
[33] "read_dtm_MC"             "readDataframe"
[35] "readDOC"                 "reader"
[37] "readPDF"                 "readPlain"
[39] "readRCV1"                "readRCV1asPlain"
[41] "readReut21578XML"        "readReut21578XMLasPlain"
[43] "readTagged"              "readXML"
[45] "removeNumbers"           "removePunctuation"
[47] "removeSparseTerms"       "removeWords"
[49] "scan_tokenizer"          "SimpleCorpus"
[51] "SimpleSource"            "stemCompletion"
[53] "stemDocument"            "stepNext"
```

```
[55] "stopwords"               "stripWhitespace"
[57] "TermDocumentMatrix"      "termFreq"
[59] "Terms"                   "tm_filter"
[61] "tm_index"                "tm_map"
[63] "tm_parLapply"            "tm_parLapply_engine"
[65] "tm_reduce"               "tm_term_score"
[67] "URISource"               "VCorpus"
[69] "VectorSource"            "weightBin"
[71] "WeightFunction"          "weightSMART"
[73] "weightTf"                "weightTfIdf"
[75] "writeCorpus"             "XMLSource"
[77] "XMLTextDocument"         "Zipf_plot"
[79] "ZipSource"
```

- Text documents are processed at different levels:
  1. **Strings** like "Hello world"
  2. **Documents** like a text of many strings stored as vector, dataframe
  3. **Corpora** as collections of documents
- The main purpose of these packages is to clean large bodies of diverse documents in preparation for more advanced analysis.

Creating a vector source

- Let's get some text first:
  1. remove `text` from the R objects list
  2. read a CSV file into a header-less data frame
  3. transpose the data frame (columns become rows)
  4. turn transposed data frame into vector

```
rm(text)   # remove the old text vector (if it exists, otherwise: warning)
read.csv(
  file="https://raw.githubusercontent.com/birkenkrahe/ds2/main/data/tm.csv",
  header=FALSE) -> text
as.vector(t(text)) -> text
str(text)
text
```

```
 chr [1:3] "Machine learning will degrade our science and debase our ethics by inc
[1] "Machine learning will degrade our science and debase our ethics by incorporat
[2] "If you want to learn R, learn the packages in this cheat sheet. These are my
[3] "BOOM! Our Free 'All Access Pass' Is Now Available! Hedgeye is the firm that's
```

- Bonus question: can you prevent the system warning in the previous code block in case there is no `text` vector present in the environment, and produce your own *personalized* warning message?

```
if (any(ls()=="text")) {
  rm(text)
} else {
  warning(
    "There is no 'text' vector in the session in \n",
    getwd())
}
```

- Use `VectorSource` to create a *source* from the `text` vector, and show its structure with `str`:

```
if (!any(search()=='package:tm')) library(tm)
source <- VectorSource(text)
str(source)
```

```
Classes 'VectorSource', 'SimpleSource', 'Source'  hidden list of 5
 $ encoding: chr ""
 $ length  : int 1
 $ position: num 0
 $ reader  :function (elem, language, id)
 $ content :function (x, ...)
```

- The source `doc_source` is a `list` of five elements and an attribute:

  1. `encoding` says that the content is encoded with apostrophs.
  2. `length = 3` is the length of the input vector
  3. `position = 0` means that there is no other document in the source
  4. `reader` is the function used to process the vector
  5. `content` is the content of the corpus - one string only

6. `attr` is a vector that says what type of source this is

```
typeof(source)
```

```
[1] "list"
```

Creating a volatile corpus

- To turn the `VectorSource` into a volatile (in-memory) corpus, use `VCorpus` (that's also a `list`):

```
corpus <- VCorpus(VectorSource(text))
corpus
typeof(corpus)
str(corpus)
```

```
<<VCorpus>>
Metadata:  corpus specific: 0, document level (indexed): 0
Content:  documents: 3
[1] "list"
List of 3
 $ 1:List of 2
  ..$ content: chr ""
  ..$ meta   :List of 7
  .. ..$ author       : chr(0)
  .. ..$ datetimestamp: POSIXlt[1:1], format: "2023-04-01 23:55:37"
  .. ..$ description  : chr(0)
  .. ..$ heading      : chr(0)
  .. ..$ id           : chr "1"
  .. ..$ language     : chr "en"
  .. ..$ origin       : chr(0)
  .. ..- attr(*, "class")= chr "TextDocumentMeta"
  ..- attr(*, "class")= chr [1:2] "PlainTextDocument" "TextDocument"
 $ 1:List of 2
  ..$ content: chr ""
  ..$ meta   :List of 7
  .. ..$ author       : chr(0)
  .. ..$ datetimestamp: POSIXlt[1:1], format: "2023-04-01 23:55:37"
  .. ..$ description  : chr(0)
```

```
  .. ..$ heading      : chr(0)
  .. ..$ id           : chr "1"
  .. ..$ language     : chr "en"
  .. ..$ origin       : chr(0)
  .. ..- attr(*, "class")= chr "TextDocumentMeta"
  ..- attr(*, "class")= chr [1:2] "PlainTextDocument" "TextDocument"
 $ 1:List of 2
  ..$ content: chr [1:2] "UseMethod" "text"
  ..$ meta   :List of 7
  .. ..$ author       : chr(0)
  .. ..$ datetimestamp: POSIXlt[1:1], format: "2023-04-01 23:55:37"
  .. ..$ description  : chr(0)
  .. ..$ heading      : chr(0)
  .. ..$ id           : chr "1"
  .. ..$ language     : chr "en"
  .. ..$ origin       : chr(0)
  .. ..- attr(*, "class")= chr "TextDocumentMeta"
  ..- attr(*, "class")= chr [1:2] "PlainTextDocument" "TextDocument"
 - attr(*, "class")= chr [1:2] "VCorpus" "Corpus"
```

- A corpus can have metadata - this only only has two "documents", i.e. the two strings. A corpus can have any number of documents.
- You can inspect the corpus with `tm::inspect`. This provides information about each of the documents -

```
inspect(corpus)

<<VCorpus>>
Metadata:  corpus specific: 0, document level (indexed): 0
Content:  documents: 3

$x
<<PlainTextDocument>>
Metadata:  7
Content:  chars: 0

$...
<<PlainTextDocument>>
Metadata:  7
```

```
Content:  chars: 0

[[3]]
<<PlainTextDocument>>
Metadata:  7
Content:  chars: 13
```

- Individual documents can be accessed with the `[[` operator or via their name:

```
meta(corpus[[3]]) # metadata for document no. 1 (list index)
meta(corpus[[3]],"language") # metadata for document language
```

```
  author       : character(0)
  datetimestamp: 2023-04-01 23:55:37
  description  : character(0)
  heading      : character(0)
  id           : 1
  language     : en
  origin       : character(0)
[1] "en"
```

- Accessing the corpus document content with `content`:

```
content(corpus[[1]])
corpus[[1]][1]
as.character(corpus[[1]])
```

```
[1] ""
$content
[1] ""
[1] ""
```

- You can also make a corpus from a data frame and store it permanently in a database using the `PCorpus` function.

TM Function	Description	Before	After
`tolower()`	Makes all text lowercase	`Starbucks is from Seattle.`	`starbucks is from seattle.`
`removePunctuation()`	Removes punctuation like periods and exclamation points	`Watch out! That coffee is going to spill!`	`Watch out That coffee is going to spill`
`removeNumbers()`	Removes numbers	`I drank 4 cups of coffee 2 days`	`I drank cups of coffee days ago.`
`stripWhiteSpace()`	Removes tabs and extra spaces	`I like coffee.`	`I like coffee.`
`removeWords()`	Removes specific words (e.g. "the", "of") defined by the data scientist	`The coffee house and barista he visited were nice, she said hello.`	`The coffee house barista visited nice, said hello.`

Figure 1: Text mining functions

Cleaning a string

- Base R cleaning functions in `tm` and base R:
- The function `tolower` is actually a `base R` function:
  1. check out the namespace of `tolower` with `environment`
  2. print the first message of the `corpus` with `content`
  3. apply `tolower` to the first message in our `corpus`

```
environment(tolower)
content(corpus[[1]])
tolower(content(corpus[[1]]))
tolower(corpus[[1]])
```

```
<environment: namespace:base>
[1] ""
[1] ""
[1] ""
```

- Achieve the last result using a pipeline with the `|>` operator:

```
   corpus[[1]] |>
#     content() |>
     tolower()

[1] ""
```

- Save the 2nd `corpus` document in an object `t`, then use the following functions (in this order) on `t` and save the result in `tc`:

  1. `removeWords(t,stopwords("en"))`
  2. `removeNumbers`
  3. `removePunctuation`
  4. `stripWhitespace`
  5. `tolower`

```
content(corpus[[2]]) -> t
t
tolower(
  stripWhitespace(
    removePunctuation(
      removeNumbers(
        removeWords(t, stopwords("en")))))) -> tc
tc

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

- Here, `stopwords` is a function, and `stopwords("en")` is a dictionary of English "small" words to be removed:

```
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 if the words "good" and "at" are in the English stop words dictionary:

```
any(stopwords("en")==c("at"))
any(stopwords("en")==c("good"))
"good" %in% stopwords("en")
"at" %in% stopwords("en")
```

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

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

- Why is "good" not a stop word?
- Recreate the cleaning from before using a pipeline:

```
content(corpus[[2]]) -> t
t |>
  removeWords(stopwords("en")) |>
  removeNumbers() |>
  removePunctuation() |>
  stripWhitespace() |>
  tolower()
```

```
[1] ""
```

- The qdap package contains even more cleaning functions. Check the methods in the package:

```
library(qdap)
ls('package:qdap')
```

```
  [1] "%&%"                          "%>%"
  [3] "%bs%"                         "%ex%"
  [5] "%sw%"                         "add_incomplete"
  [7] "add_s"                        "adjacency_matrix"
  [9] "adjmat"                       "all_words"
 [11] "Animate"                      "apply_as_df"
 [13] "apply_as_tm"                  "as.Corpus"
 [15] "as.DocumentTermMatrix"        "as.dtm"
 [17] "as.tdm"                       "as.TermDocumentMatrix"
 [19] "as.wfm"                       "automated_readability_index"
 [21] "bag_o_words"                  "beg2char"
 [23] "blank2NA"                     "boolean_search"
 [25] "bracketX"                     "bracketXtract"
 [27] "breaker"                      "build_qdap_vignette"
 [29] "capitalizer"                  "char_table"
 [31] "char2end"                     "character_count"
 [33] "character_table"              "check_spelling"
```

```
 [35] "check_spelling_interactive"  "check_text"
 [37] "chunker"                     "clean"
 [39] "cm_2long"                    "cm_code.blank"
 [41] "cm_code.combine"             "cm_code.exclude"
 [43] "cm_code.overlap"             "cm_code.transform"
 [45] "cm_combine.dummy"            "cm_df.fill"
 [47] "cm_df.temp"                  "cm_df.transcript"
 [49] "cm_df2long"                  "cm_distance"
 [51] "cm_dummy2long"               "cm_long2dummy"
 [53] "cm_range.temp"               "cm_range2long"
 [55] "cm_time.temp"                "cm_time2long"
 [57] "colcomb2class"               "coleman_liau"
 [59] "colpaste2df"                 "colSplit"
 [61] "colsplit2df"                 "combo_syllable_sum"
 [63] "comma_spacer"                "common"
 [65] "condense"                    "correct"
 [67] "counts"                      "cumulative"
 [69] "DATA"                        "DATA.SPLIT"
 [71] "DATA2"                       "delete"
 [73] "dir_map"                     "discourse_map"
 [75] "dispersion_plot"             "Dissimilarity"
 [77] "dist_tab"                    "diversity"
 [79] "duplicates"                  "edge_apply"
 [81] "end_inc"                     "end_mark"
 [83] "end_mark_by"                 "env.syl"
 [85] "exclude"                     "Filter"
 [87] "flesch_kincaid"              "folder"
 [89] "formality"                   "freq_terms"
 [91] "fry"                         "gantt"
 [93] "gantt_plot"                  "gantt_rep"
 [95] "gantt_wrap"                  "genX"
 [97] "genXtract"                   "gradient_cloud"
 [99] "hamlet"                      "htruncdf"
[101] "imperative"                  "incomp"
[103] "incomplete_replace"          "inspect_text"
[105] "is.global"                   "key_merge"
[107] "kullback_leibler"            "lcolsplit2df"
[109] "left_just"                   "lexical_classification"
[111] "linsear_write"               "ltruncdf"
[113] "lview"                       "mcsv_r"
```

```
[115] "mcsv_w"                  "mgsub"
[117] "mraja1"                  "mraja1spl"
[119] "multigsub"               "multiscale"
[121] "NAer"                    "name2sex"
[123] "Network"                 "new_project"
[125] "ngrams"                  "object_pronoun_type"
[127] "outlier_detect"          "outlier_labeler"
[129] "paste2"                  "phrase_net"
[131] "plot_gantt_base"         "polarity"
[133] "polysyllable_sum"        "pos"
[135] "pos_by"                  "pos_tags"
[137] "potential_NA"            "preprocessed"
[139] "pres_debate_raw2012"     "pres_debates2012"
[141] "pronoun_type"            "prop"
[143] "proportions"             "qcombine"
[145] "qcv"                     "qdap_df"
[147] "qheat"                   "qprep"
[149] "qtheme"                  "question_type"
[151] "qview"                   "raj"
[153] "raj.act.1"               "raj.act.1POS"
[155] "raj.act.2"               "raj.act.3"
[157] "raj.act.4"               "raj.act.5"
[159] "raj.demographics"        "rajPOS"
[161] "rajSPLIT"                "random_data"
[163] "random_sent"             "rank_freq_mplot"
[165] "rank_freq_plot"          "raw.time.span"
[167] "read.transcript"         "replace_abbreviation"
[169] "replace_contraction"     "replace_number"
[171] "replace_ordinal"         "replace_symbol"
[173] "replacer"                "right_just"
[175] "rm_empty_row"            "rm_row"
[177] "rm_stop"                 "rm_stopwords"
[179] "sample.time.span"        "scores"
[181] "scrubber"                "Search"
[183] "sent_detect"             "sent_detect_nlp"
[185] "sentCombine"             "sentiment_frame"
[187] "sentSplit"               "SMOG"
[189] "space_fill"              "spaste"
[191] "speakerSplit"            "stem_words"
[193] "stem2df"                 "stemmer"
```

```
[195] "strip"                       "strWrap"
[197] "sub_holder"                  "subject_pronoun_type"
[199] "syllable_count"              "syllable_sum"
[201] "syn"                         "syn_frame"
[203] "synonyms"                    "synonyms_frame"
[205] "term_match"                  "termco"
[207] "termco_c"                    "termco_d"
[209] "termco2mat"                  "Text"
[211] "Text<-"                      "theme_badkitchen"
[213] "theme_cafe"                  "theme_duskheat"
[215] "theme_grayscale"             "theme_greyscale"
[217] "theme_hipster"               "theme_nightheat"
[219] "theme_norah"                 "Title"
[221] "Title<-"                     "TOT"
[223] "tot_plot"                    "trans_cloud"
[225] "trans_context"               "trans_venn"
[227] "Trim"                        "truncdf"
[229] "type_token_ratio"            "unbag"
[231] "unique_by"                   "vertex_apply"
[233] "visual"                      "wc"
[235] "weight"                      "wfdf"
[237] "wfm"                         "wfm_combine"
[239] "wfm_expanded"                "which_misspelled"
[241] "word_associate"              "word_cor"
[243] "word_count"                  "word_diff_list"
[245] "word_length"                 "word_list"
[247] "word_network_plot"           "word_position"
[249] "word_proximity"              "word_split"
[251] "word_stats"

## save.image("../data/ds2_20230331")
save.image(".RData")
shell("DIR .RData")

 Volume in drive C is OS
 Volume Serial Number is 0654-135C

 Directory of c:\Users\birkenkrahe\Documents\GitHub\ds2\org

04/01/2023  06:55 PM            14,085 .RData
```

```
               1 File(s)         14,085 bytes
               0 Dir(s)  143,044,263,936 bytes free
```

- Load when you come back to this analysis:

```
load("../data/ds2_20230331")
search()
ls()
```

```
 [1] ".GlobalEnv"               "package:qdap"
 [3] "package:RColorBrewer"     "package:qdapTools"
 [5] "package:qdapRegex"        "package:qdapDictionaries"
 [7] "package:tm"               "package:NLP"
 [9] "ESSR"                     "package:stats"
[11] "package:graphics"         "package:grDevices"
[13] "package:utils"            "package:datasets"
[15] "package:stringr"          "package:httr"
[17] "package:methods"          "Autoloads"
[19] "package:base"
 [1] "A"           "api_key"     "ask_chatgpt" "B"           "bar"
 [6] "baz"         "C"           "corpus"      "D"           "foo"
[11] "hello"       "hello_"      "hello_1"     "mult1"       "mult2"
[16] "mult2_"      "mult3"       "mult3_"      "multiply"    "multiply_"
[21] "myfibplot"   "myfibplot_"  "source"      "t"           "tc"
[26] "term_count"  "text"        "unpackme"    "x"
```

READ Using `gsub` and `tm::removePunctuation`

Cleaning a corpus

- To clean a corpus (a collection of different documents), use `tm_map`, which works as a wrapper. For example for `removePunctuation` and our `corpus`:

```
library(tm)
nchar(content(corpus[[3]]))
nchar(content(tm_map(corpus, removePunctuation)[[3]]))
nchar(content(tm_map(corpus, removeWords, words=stopwords("en"))[[3]]))
nchar(content(tm_map(corpus, content_transformer(tolower))[[3]]))
```

The `removePunctuation()` transformation completely strips punctuation characters from the text, which can lead to unintended consequences. For example, consider what happens when it is applied as follows:

```
> removePunctuation("hello...world")
[1] "helloworld"
```

As shown, the lack of a blank space after the ellipses caused the words *hello* and *world* to be joined as a single word. While this is not a substantial problem right now, it is worth noting for the future.

Tip

To work around the default behavior of `removePunctuation()`, create a custom function that replaces rather than removes punctuation characters:

```
> replacePunctuation <- function(x) {
    gsub("[[:punct:]]+", " ", x)
}
```

Tip

This uses R's `gsub()` function to substitute any punctuation characters in `x` with a blank space. This `replacePunctuation()` function can then be used with `tm_map()` as with other transformations.

Figure 2: Source: Lantz, Machine learning with R (2e,2019)

```
[1] 269
[1] 252
[1] 250
[1] 269
```

- Bonus: we only have 3 strings in the corpus, so an index 4 will be out of bounds. How can you make the first command safe against this error?

```
library(tm)
length(content(corpus)) -> b
for (i in 1:4) {
  if (i > b) {
    stop("Index out of bounds: only ",b,
         " elements exist.\nCommand terminated.")
  } else {
    print(nchar(content(corpus[[i]])))
  }
}
```

```
[1] 161
[1] 134
[1] 269
Error: Index out of bounds: only 3 elements exist.
Command terminated.
```

Creating a Term-Document-Matrix (TDM)

- Bag-of-words only cares about term (aka word) frequencies - this information is contained in a Term-Document-Matrix whose rows are terms and whose columns are the indidivual documents of the corpus.
- The function `clean_corpus` has been defined and contains all the cleaning operations you've seen so far:
  1. run `clean_corpus` on `corpus` and save in object `clean_corpus`
  2. print element 2 of `clean_corpus`

```
clean_corpus <- function(corpus) {
```

	Tweet 1	Tweet 2	Tweet 3	...	Tweet N
Term 1	0	0	0	0	0
Term 2	1	1	0	0	0
Term 3	1	0	0	0	0
...	0	0	3	1	1
Term M	0	0	0	1	0

Term Document Matrix (TDM)

	Term 1	Term 2	Term 3	...	Term M
Tweet 1	0	1	1	0	0
Tweet 2	0	1	0	0	0
Tweet 3	0	0	0	3	0
...	0	0	0	1	1
Tweet N	0	0	0	1	0

Document Term Matrix (DTM)

Figure 3: TDM and DTM for a corpus of tweets.

```
  corpus <- tm_map(corpus,
                   removeNumbers)
  corpus <- tm_map(corpus,
                   removePunctuation)
  corpus <- tm_map(corpus,
                   content_transformer(tolower))
  corpus <- tm_map(corpus,
                   removeWords,
                   words = c(stopwords("en")))
  corpus <- tm_map(corpus,
                   stripWhitespace)
  return(corpus)
}
clean_corpus(corpus) -> clean_corpus
content(clean_corpus[[2]])
```

```
[1] " want learn r learn packages cheat sheet tools rstats datascience httpsbuffly
```

- Notice that the order of operations matters a lot for a truly "clean" result. For example, applying `tolower` after `removeWords` will leave "If" because the dictionary only contains "if".

- The `tm::TermDocumentMatrix~` function turns the `clean_corpus` into a TDM:

```
tdm <- TermDocumentMatrix(clean_corpus)
tdm
```

```
<<TermDocumentMatrix (terms: 51, documents: 3)>>
Non-/sparse entries: 51/102
Sparsity           : 67%
Maximal term length: 16
Weighting          : term frequency (tf)
```

- Look at the structure - you can see that the column vector names contain the term and document information:

```
str(tdm)

List of 6
 $ i       : int [1:51] 6 9 10 11 14 16 23 27 28 30 ...
 $ j       : int [1:51] 1 1 1 1 1 1 1 1 1 1 ...
 $ v       : num [1:51] 1 1 1 1 1 1 1 1 1 1 ...
 $ nrow    : int 51
 $ ncol    : int 3
 $ dimnames:List of 2
  ..$ Terms: chr [1:51] "access" "available" "boom" "called" ...
  ..$ Docs : chr [1:3] "1" "2" "3"
 - attr(*, "class")= chr [1:2] "TermDocumentMatrix" "simple_triplet_matrix"
 - attr(*, "weighting")= chr [1:2] "term frequency" "tf"
```

- Transpose the TDM to a DTM using `base::t` (or use `DocumentTermMatrix` on the clean corpus):

```
dtm <- t(tdm)
dtm
tdm

<<DocumentTermMatrix (documents: 3, terms: 51)>>
Non-/sparse entries: 51/102
Sparsity           : 67%
Maximal term length: 16
Weighting          : term frequency (tf)
<<TermDocumentMatrix (terms: 51, documents: 3)>>
Non-/sparse entries: 51/102
Sparsity           : 67%
Maximal term length: 16
Weighting          : term frequency (tf)
```

Analyze and visualize the TDM

- All we're interested in, and all we can analyze and visualize, are term frequencies.
- To see counts, you can transform the TDM into a matrix:

```
as.matrix(tdm) -> m
head(m, 10)
```

```
     Docs
Terms         1 2 3
  access      0 0 1
  available   0 0 1
  boom        0 0 1
  called      0 0 1
  cheat       0 1 0
  conception  1 0 0
  crash       0 0 1
  datascience 0 1 0
  debase      1 0 0
  degrade     1 0 0
```

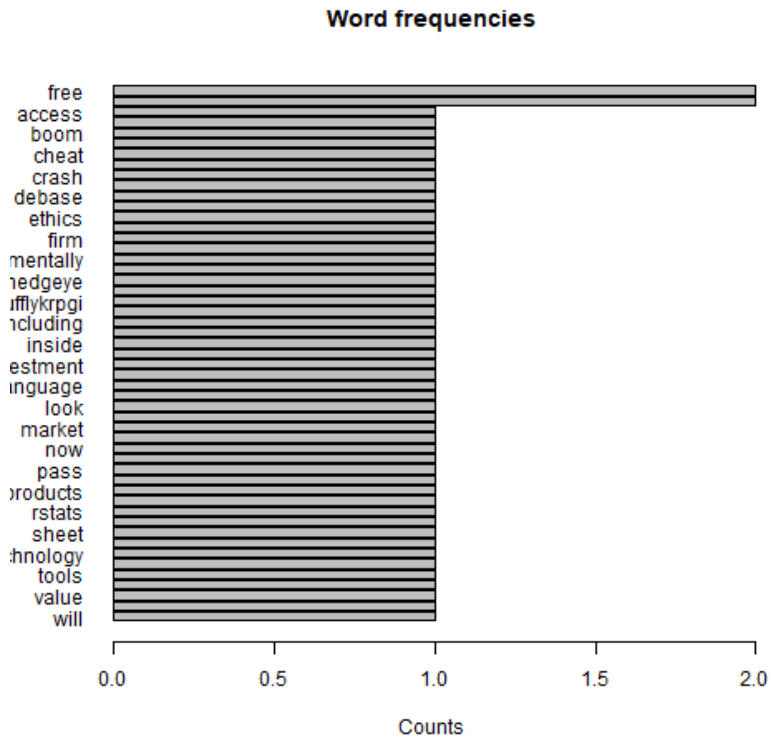
- To see top counts:
  1. sum over all documents and get the frequencies for each term
  2. sort the entries in decreasing order
  3. print the top six entries

```
rowSums(m) -> freq
sort(freq, decreasing=TRUE) -> sorted
head(sorted)
```

```
free     learn    access available      boom    called
   2         2         1         1         1         1
```

- You can visualize the results as a barchart or as a wordcloud. For the wordclouds, we need the `wordcloud` package.
- Barchart:

```
barplot(rev(sorted),
        horiz=TRUE,
        main="Word frequencies",
        xlab="Counts",
        las=1)
```



- For the wordcloud, we transform the sorted, named frequency vector `sorted` into a dataframe and then remove the `rownames`:

```
library(wordcloud)
df <- data.frame(term=names(sorted),
                 num=sorted)
rownames(df) <- NULL
head(df,10)

Warning message:
```

```
package 'wordcloud' was built under R version 4.2.3
  term num
1         free   2
2        learn   2
3       access   1
4    available   1
5         boom   1
6       called   1
7        cheat   1
8   conception   1
9        crash   1
10 datascience   1
```

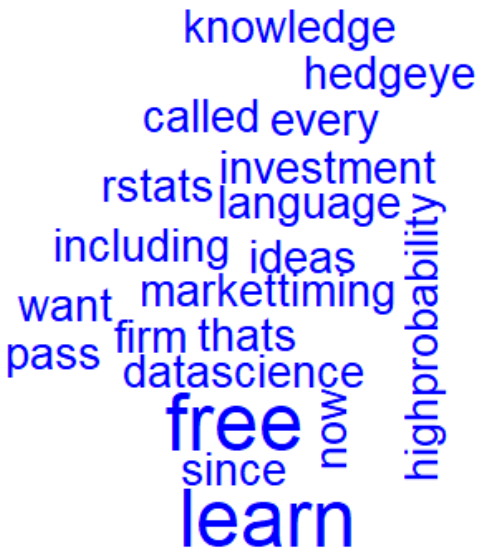
- Now we apply the `wordcloud` function, which requires words (`term`), and frequencies (`freq`). Check the arguments of this function:

```
args(wordcloud)
```

```
function (words, freq, scale = c(4, 0.5), min.freq = 3, max.words = Inf,
    random.order = TRUE, random.color = FALSE, rot.per = 0.1,
    colors = "black", ordered.colors = FALSE, use.r.layout = FALSE,
    fixed.asp = TRUE, ...)
NULL
```

- Create the word cloud:

```
wordcloud(words = df$term,
          freq = df$num,
          max.words=20,
          color="blue")
```



Resources

- Cleaning function for `corpus`:

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

```
  return(corpus)
}
```