Text mining in practice - Bag of Words - Intro to word clouds

Digital Humanities DSC 105 Spring 2023

Marcus Birkenkrahe

March 30, 2023

README

- This lecture closely follows the DataCamp lesson "Text Mining with Bag-of-Words in R" by Ted Kwartler, chapter 2, lesson 2, "Intro to word clouds" (Link).
- Download and open the practice file `8_wordclouds_practice.org` from GitHub to code along.
- In this lecture & practice:
    1. simple word cloud creation using Chardonnay tweets
    2. adding stop words to change word cloud
    3. improve word cloud appearance and insights

Get the corpus data and the R packages

- Download `corpora.R` from GitHub (bit.ly/tm-corpora)
- Run the file on the shell (`M-x eshell`) as a batch job:

```
R CMD BATCH corpora.R
ls -al .RData
```

- Load the `.RData` file in your current R session and check that packages and user-defined objects were loaded:

```
load_packages <- function() {
    library(tm)
    library(qdap)
    library(SnowballC)
    library(wordcloud)
    search()
}
load_packages()
load("c:/Users/birkenkrahe/Downloads/.RData")
search()
ls()
```

```
 [1] ".GlobalEnv"               "package:viridisLite"
 [3] "package:wordcloud"        "package:SnowballC"
 [5] "package:qdap"             "package:RColorBrewer"
 [7] "package:qdapTools"        "package:qdapRegex"
 [9] "package:qdapDictionaries" "package:tm"
[11] "package:NLP"              "ESSR"
[13] "package:stats"            "package:graphics"
[15] "package:grDevices"        "package:utils"
[17] "package:datasets"         "package:stringr"
[19] "package:httr"             "package:methods"
[21] "Autoloads"                "package:base"
 [1] ".GlobalEnv"               "package:viridisLite"
 [3] "package:wordcloud"        "package:SnowballC"
 [5] "package:qdap"             "package:RColorBrewer"
 [7] "package:qdapTools"        "package:qdapRegex"
 [9] "package:qdapDictionaries" "package:tm"
[11] "package:NLP"              "ESSR"
[13] "package:stats"            "package:graphics"
[15] "package:grDevices"        "package:utils"
[17] "package:datasets"         "package:stringr"
[19] "package:httr"             "package:methods"
[21] "Autoloads"                "package:base"
 [1] "api_key"                 "ask_chatgpt"
 [3] "chardonnay_corpus"       "chardonnay_df"
 [5] "chardonnay_m"            "chardonnay_src"
 [7] "chardonnay_tdm"          "chardonnay_vec"
 [9] "clean_chardonnay"        "clean_chardonnay_corpus"
[11] "clean_coffee"            "clean_coffee_corpus"
```

```
[13] "coffee_corpus"          "coffee_df"
[15] "coffee_m"               "coffee_src"
[17] "coffee_tdm"             "coffee_vec"
[19] "color_pal"              "idx"
[21] "load_packages"          "m"
[23] "M"                      "max"
[25] "stops"                  "term_frequency"
[27] "terms"                  "word_freq"
```

- You need the `clean_coffee` and `clean_chardonnay` corpora.
- If we don't finish with a session, save your data from now on:

```
save.image(file=".RData")
shell("ls -al .RData")
```

```
-rwx------+ 1 Birkenkrahe LYONNET+Group(513) 755072 Mar 30 08:53 .RData
```

Intro to word clouds

- Our starting point is the cleaned corpus of "Chardonnay" tweets, `clean_chardonnay`.
- Look at the corpus:

```
clean_chardonnay
```

```
<<VCorpus>>
Metadata:  corpus specific: 0, document level (indexed): 0
Content:  documents: 1000
```

Convert TDM to matrix

- Convert Chardonnay TDM to a matrix and check its dimensions:

```
library(tm)
chardonnay_tdm <- TermDocumentMatrix(clean_chardonnay)
chardonnay_m <- as.matrix(chardonnay_tdm)
dim(chardonnay_m)
```

```
[1] 3067 1000
```

Create the frequency table

- The starting point of any visualization is a frequency table - it only has two columns: terms (`term`) and their counts (`num`).
- Sum rows and sort by frequency:

```
term_frequency <- rowSums(chardonnay_m)
term_frequency <- sort(term_frequency, decreasing=TRUE)
word_freq <- data.frame(term = names(term_frequency),
                        num = term_frequency)
rownames(word_freq) <- NULL
head(word_freq,n=10)
```

```
 term num
1  chardonnay 822
2         amp 120
3      marvin 104
4        wine  83
5        gaye  76
6        just  75
7       glass  63
8        like  55
9      bottle  47
10        lol  43
```

Add stop words and re-run the cleaning code

- The words `amp`, `wine` and `glass` do not help much - how can we get rid of them at this stage of our investigation? Do you know what "amp" means in this context?[1]

  Answer:

  1. download the latest version of `corpora.R` from GitHub.
  2. add these words to the stopwords cleaning function in `corpora.R`

[1] Funnily enough, I had no idea until I looked into the raw `CSV` file: `amp` is a remnant of `&amp` after `removePunctuation`, and it's the HTML short code for `&`, which is frequent in tweets (saves 2 letters). As an interesting aside: I am already so dependent on ChatGPT that instead of checking the data, I went and asked the bot about "amp in the context of Chardonnay" but to no avail, of course.

3. run the batch job with `R CMD BATCH`
4. re-load `.RData` in this file.

You'll see that the number of words (records) has gone down and the list of top frequency words is changed.

- After cleaning out the additional words, reload the data, create the TDM and the word frequency data frame:

```
load("~/Downloads/.RData")
library(tm)
chardonnay_tdm <- TermDocumentMatrix(clean_chardonnay)
chardonnay_m <- as.matrix(chardonnay_tdm)
dim(chardonnay_m)
term_frequency <- rowSums(chardonnay_m)
term_frequency <- sort(term_frequency, decreasing=TRUE)
word_freq <- data.frame(term = names(term_frequency),
                        num = term_frequency)
rownames(word_freq) <- NULL
head(word_freq,n=10)
```

```
[1] 3067 1000
 term num
1  chardonnay 822
2         amp 120
3      marvin 104
4        wine  83
5        gaye  76
6        just  75
7       glass  63
8        like  55
9      bottle  47
10        lol  43
```

Using the `wordcloud` function

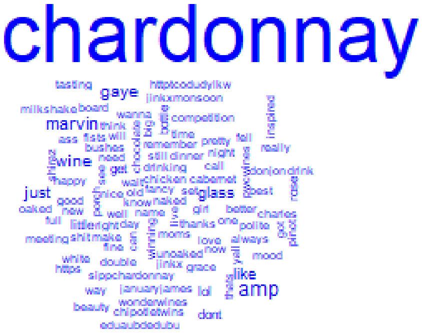
- We want to create word clouds. Is there a `wordcloud` function in `tm` or `qdap` or `base`? How can you find out? Load these packages (again, just in case) and check each of them for the function:

```
library(tm)
library(qdap)
library(wordcloud)
any(ls('package:tm')=="wordcloud")
any(ls('package:qdap')=="wordcloud")
any(ls('package:wordcloud')=="wordcloud")
```

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

- To create a wordcloud, use the `wordcloud` function. Look at the `help`.
- Use the column vectors `term` and `num` for the `words` and `freq` parameters, respectively:

```
library(wordcloud)
wordcloud(words=word_freq$term,
          freq=word_freq$num,
          max.words=100,
          color="blue")
```



- Print out frirst 10 entries of `term_frequency`:

```
term_frequency[1:10]
```

```
chardonnay        amp     marvin       wine       gaye       just      glass
       822        120        104         83         76         75         63
      like     bottle        lol
        55         47         43
```

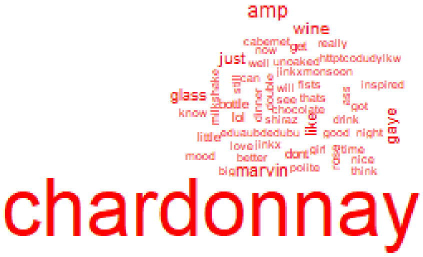
- Extract the terms 2 to 11 using `names` on `term_frequency` and call the vector of strings `terms_vec`. Show the entries 2 to 11:

```
terms_vec <- names(term_frequency)
terms_vec[2:11]
length(terms_vec)
head(table(term_frequency))
```

```
 [1] "amp"    "marvin" "wine"   "gaye"   "just"   "glass"  "like"   "bottle"
 [9] "lol"    "little"
[1] 3067
term_frequency
   1    2    3    4    5    6
1978  490  179  133   54   38
```

- Create a wordcloud using `term_vec` as the words, and `term_frequency` (defined earlier before creating the data frame `word_freq`) as the values. Add `max.words=50` and `colors="red"`:

  ```
  wordcloud(words=terms_vec,
            freq=term_frequency,
            max.words=50,
            colors="red")
  ```



- Review a cleaned tweet: do you remember how to index corpus tweets?

```
content(clean_chardonnay[[24]])

[1] " brought marvin gaye chardonnay"
```

- You can add to the stopwords, and run `tm_map` with `removeWords` on the clean corpus to remove additional words:

```
content(clean_chardonnay[[24]])
stops <- c(stopwords("en"), 'just','like')
tail(stops)
clean_chardonnay_corpus <- tm_map(clean_chardonnay,
                                  removeWords,
                                  stops)
content(clean_chardonnay_corpus[[24]])

[1] " brought marvin gaye chardonnay"
[1] "so"   "than" "too"  "very" "just" "like"
[1] " brought marvin gaye chardonnay"
```

- To see the updated word cloud, re-run the code chunks from before with the new, cleaner corpus, then go back and rerun the last plot:

```
clean_chardonnay <- clean_chardonnay_corpus
library(tm)
chardonnay_tdm <- TermDocumentMatrix(clean_chardonnay)
chardonnay_m <- as.matrix(chardonnay_tdm)
dim(chardonnay_m)
term_frequency <- rowSums(chardonnay_m)
term_frequency <- sort(term_frequency, decreasing=TRUE)
word_freq <- data.frame(term = names(term_frequency),
                        num = term_frequency)
rownames(word_freq) <- NULL
head(word_freq,n=10)

[1] 3065 1000
 term num
1  chardonnay 822
2         amp 120
3      marvin 104
```

```
4        wine  83
5        gaye  76
6       glass  63
7      bottle  47
8         lol  43
9      little  35
10       rose  34
```

Improve word clouds with different colors

- The available colors are stored in a `character` vector `colors()`. Look at the `head` of the vector, and verify that 657 colors are available:

  ```
  head(colors())
  length(colors())
  ```

  ```
  [1] "white"         "aliceblue"     "antiquewhite"  "antiquewhite1"
  [5] "antiquewhite2" "antiquewhite3"
  [1] 657
  ```

- Instead of coloring all words with the same color, you can assign a vector of different colors to `wordcloud` to make certain words stand out or to fit a specific color scheme (e.g. to accommodate color-blind people).

- If you look at the `wordcloud` arguments:

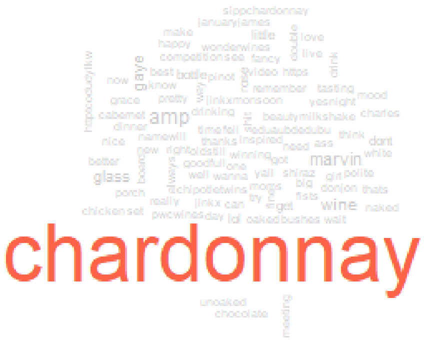
  ```
  load_packages <- function() {
      library(tm)
      library(qdap)
      library(SnowballC)
      library(wordcloud)
      search()
  }
  load_packages()
  args(wordcloud)
  ```

  ```
   [1] ".GlobalEnv"               "package:viridisLite"
   [3] "package:wordcloud"        "package:SnowballC"
  ```

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

- The `colors` argument colors words from least to most frequent. The code uses three colors of increasing vibrancy - this will naturally divide the term frequency into "low", "medium", and "high":

```
term_frequency <- rowSums(chardonnay_m)
term_frequency <- sort(term_frequency, decreasing=TRUE)
word_freq <- data.frame(term = names(term_frequency),
                        num = term_frequency)
rownames(word_freq) <- NULL
head(word_freq,n=10)
wordcloud(words=word_freq$term,
          freq=word_freq$num,
          max.words=100,
          colors=c("grey80","darkgoldenrod1","tomato"))
```



```
str(word_freq)

'data.frame': 3065 obs. of  2 variables:
 $ term: chr  "chardonnay" "amp" "marvin" "wine" ...
 $ num : num  822 120 104 83 76 63 47 43 35 34 ...
```

Using prebuilt color palettes: `viridisLite`

- The `viridisLite` package contains color maps designed to improve graph readability for readers with color vision deficiencies.
- Also, the colors translate well into black-and-white versions without loss of readability.
- Install `viridisLite` in the R console, load it and check success:

```
library(viridisLite)
search()
```

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

- Look at the contents of the package with `ls`: these are the different color maps.

  ```
  ls('package:viridisLite')
  ```

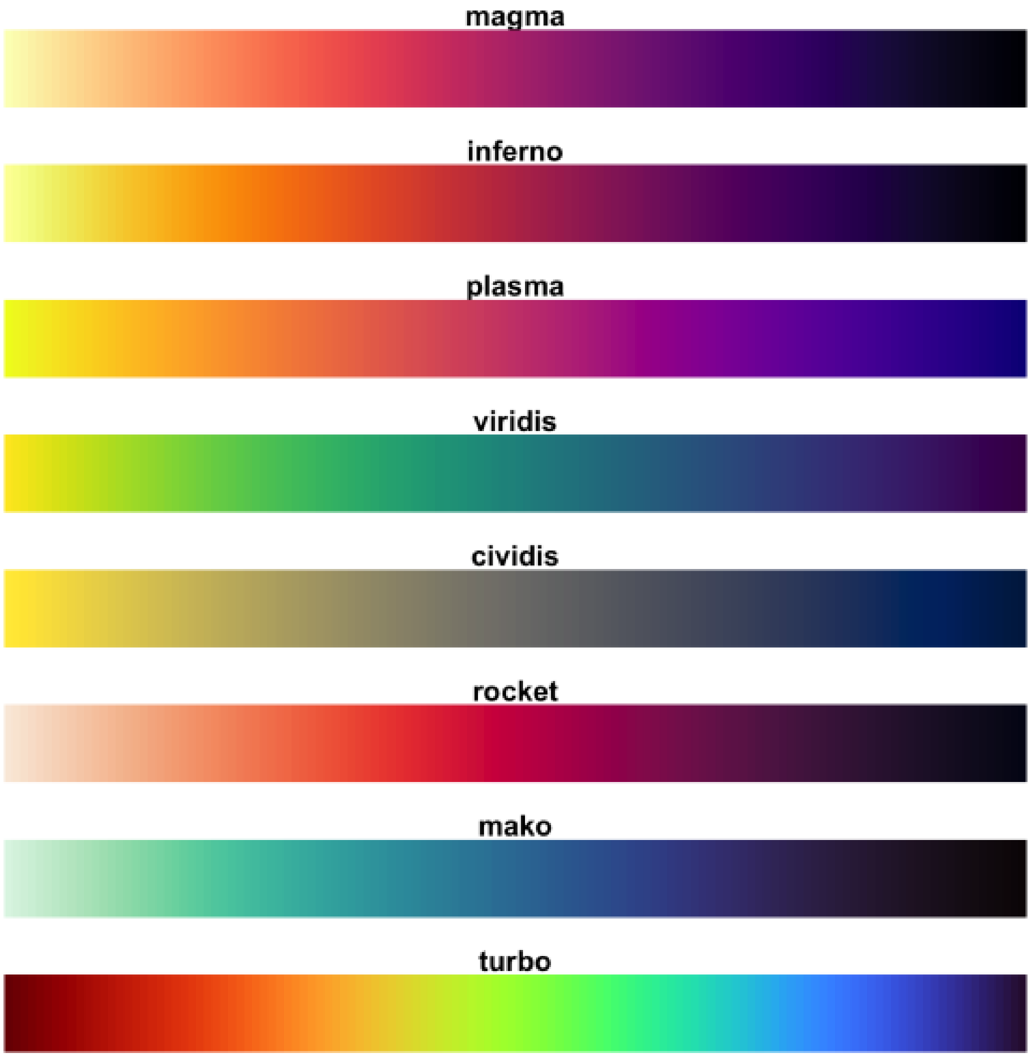
  ```
  [1] "cividis"     "inferno"     "magma"       "mako"        "plasma"
  [6] "rocket"      "turbo"       "viridis"     "viridis.map" "viridisMap"
  ```

- All maps are functions with one mandatory argument, the number of colors `n` used. Check the arguments of `viridisLite::cividis`:

  ```
  args(cividis)
  ```

  ```
  function (n, alpha = 1, begin = 0, end = 1, direction = 1)
  NULL
  ```

- As the vignette for `viridisLite` reveals, the other parameter allow to change transparency (`alpha`), hue (`begin` and `end`), and order. Here are the color scales for the maps:



- To created a new wordcloud with the selected palette, select 5 colors from `turbo` and store them in a vector `color_pal`:

  ```
  color_pal <- turbo(5)
  ```

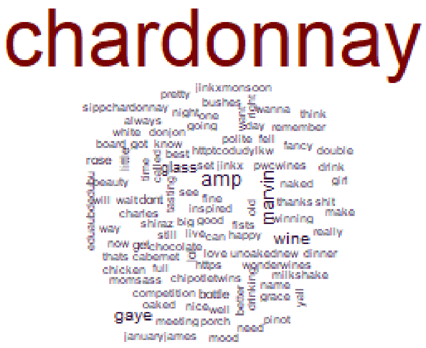
- Print the hex-codes for `color_pal` to the console:

  ```
  color_pal
  ```

```
[1] "#30123BFF" "#28BBECFF" "#A2FC3CFF" "#FB8022FF" "#7A0403FF"
```

- Create a word cloud from the Chardonnay tweets `word_freq`, include 100 terms, and set the `colors` to the `color_pal` palette:

  ```
  wordcloud(words=word_freq$term,
            freq=word_freq$num,
            max.words=100,
            colors=color_pal)
  ```



-

Load packages

```
load_packages <- function() {
    library(tm)
```

```
    library(qdap)
    library(SnowballC)
    library(wordcloud)
    search()
}
load_packages()
```

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