17 - Word Cloud

ST 597 | Spring 2017
University of Alabama

17-wordcloud.pdf

Contents

Required Packages and Data

```
library(tidyverse)
library(stringr)
library(tm)           # install.packages("tm")
library(wordcloud)    # install.packages("wordcloud")
library(SnowballC)    # install.packages("SnowballC")
library(RColorBrewer) # install.packages("RColorBrewer")
```

1 Text Mining

We are going to use some of the functions from the `tm` package to do some basic text mining and build a word cloud. The `tm` package has a vignette (https://cran.r-project.org/web/packages/tm/vignettes/tm.pdf) and I found a webpage that walks through some of the steps (https://eight2late.wordpress.com/2015/05/27/a-gentle-introduction-to-text-mining-using-r/). There are doubtless many other free sites to get you started on text mining.

1.1 Goals

We are going to analyze a set of documents related to business analytics. Specifically, we are going to break a document down into a frequency distribution of its words and examine the most frequent (and potentially the most important words).

Like all topics we have covered this semester, we are only scratching the surface of what is possible in the field of text mining and text analytics. Document clustering, author attribution, sentiment analysis, natural language processing (NLP), entity extraction, word and document networks, etc. are just some examples of where you can go with this. Hopefully, we cover enough so you can start to imagine and think about what is possible with text data.

2 Document Corpus

The first step is to create a **corpus**, or collection of documents that contain text data. The `tm` package allows a few ways to do this depending on where the documents are (in memory, in database, etc.). Here are some common sources:

function	description
`?Source`	Help for setting the source
`DirSource()`	Creates a directory source (path to document directory)
`VectorSource()`	Creates a source from vector of strings (documents)

The function `DirSource()` basically reads in all the documents from a directory and `VectorSource()` loads an existing R vector of documents. We have 26 plain text (.txt) documents. We need to read these into R, create a character vector where each element is a document, and then create the corpus.

2.1 Read in Text Documents

Here I will do this manually with a loop and `read_file()`. The data files can be found here https://raw.githubusercontent.com/mdporter/ST597/master/data/BA_skills/ba-xx.txt, where `xx` is two digits between `01-26`.

```
#- read in all documents
base_url = "https://raw.githubusercontent.com/mdporter/ST597/master/data/BA_skills/ba-"
end_url = ".txt"

docs = character(26)        # create vector of 26 blank elements
for(i in 1:26){             # for loop to set the value of i
  file_num = str_pad(i, width=2, side="left", pad="0")   # make 2 digit number
  url = str_c(base_url, file_num, end_url)
  docs[i] = read_file(url)
}
```

```r
#- example document
# docs[22]                                   # raw form
writeLines(str_wrap(docs[22], width=75)) # displayed form
#> My very simple take: Programming in R/Python both for data analysis and
#> for visualization. Equally important more or less in my view. Beyond that,
#> hands-on data set analysis. Teach people to look at data and decide the
#> best approach themselves rather than telling them which approach to take
#> and grading on their ability to do so. Manager of Analytics
```

Notice how messy some of these documents are.

2.2 Create Corpus

Next, we need to tell R the type of source, in this case a vector source, then create the corpus

```r
src = VectorSource(docs)      # source
corpus = Corpus(src)          # corpus
```

The `Corpus()` function lets you specify the type of document (e.g., plain text, pdf, word, Reuters news, etc.) and language to use. The help `?Reader` can provide some additional information. But here the default values will work for us (plain text and english language).

2.2.1 Word Counts

We are going to use the functions from the `stringr` and `dplyr` packages to find the frequency of words in our documents. Here we get two word counts:

- the `total_counts` data frame gives the total number of times a word appears in all the documents (so a word that appears more than once in a document will be counted more than once.)
- the `distinct_counts` gives the number of documents that contain the word (so a word that appears more than once in a document will only be counted once.)

2.2.1.1 Get all words into a data frame

```r
#- get the words for each document
X = str_split(docs, boundary("word"))   # list of words

#- use stack() function to make data frame
names(X) = 1:length(X)    # add names to elements of X
Y = stack(X) %>%
    rename(word=values, document=ind)   # change col names
head(Y)
#>          word document
#> 1  Obviously        1
#> 2          I        1
#> 3       know        1
#> 4       more        1
#> 5      about        1
#> 6 basketball        1
```

2.2.1.2 Get word counts

```r
#- Get frequency of words (total)
(total_counts = count(Y, word, sort=TRUE) )
#> # A tibble: 1,659 × 2
```

```
#>      word     n
#>     <chr> <int>
#> 1      to   183
#> 2     and   167
#> 3     the   157
#> 4       a   138
#> 5      of   127
#> 6      in    99
#> 7      is    89
#> 8       I    84
#> 9    that    82
#> 10  data     69
#> # ... with 1,649 more rows

#- Get frequency of word occurrence (e.g., {0,1} per document)
(distinct_counts =
  Y %>%
    group_by(word) %>%
    summarize(n=n_distinct(document)) %>%
    arrange(desc(n)))
#> # A tibble: 1,659 × 2
#>          word     n
#>         <chr> <int>
#> 1         and    26
#> 2          of    25
#> 3         the    25
#> 4          to    25
#> 5          in    24
#> 6           a    22
#> 7   Analytics    21
#> 8         are    21
#> 9           I    21
#> 10       that    21
#> # ... with 1,649 more rows
```

Notice that the most common words are uninteresting: “to”, “and”, “of”, “the”. We also have lots of numbers

```
arrange(total_counts, word)      # order alphabetically (numbers first)
#> # A tibble: 1,659 × 2
#>        word     n
#>       <chr> <int>
#> 1         1     8
#> 2        10     2
#> 3       100     1
#> 4         2    10
#> 5        20     1
#> 6   200,000     1
#> 7      2018     1
#> 8      21st     1
#> 9         3     7
#> 10       30     1
#> # ... with 1,649 more rows
```

And, consider if any of these words should be considered together?

```
filter(total_counts, str_detect(word, pattern="[Aa]naly"))
#> # A tibble: 10 × 2
#>          word     n
#>         <chr> <int>
#> 1    analytics    38
```

```
#> 2   Analytics    31
#> 3    analysis    13
#> 4    analysts     8
#> 5  analytical     6
#> 6    analytic     5
#> 7    Analysis     3
#> 8     analyst     2
#> 9     analyze     1
#> 10  analyzing     1
```

3 Transformations

Before we start our data analysis and modelling, it is often necessary to modify the text in some ways. For example, the basic step of extracting the words is one task that is usually performed. To help with this, we can also

- remove whitespace
- convert letters to same case (e.g., lowercase)
- removing punctuation
- removing *stop words*, common words that do not carry much meaning to the analysis (e.g., “an”, “a”, “the”)
- removing numbers or other non-text characters

3.1 `tm` transformations

The `tm` package provides some helpful transformation functions.

```
library(tm)
getTransformations()     # list of transformations
#> [1] "removeNumbers"      "removePunctuation" "removeWords"
#> [4] "stemDocument"       "stripWhitespace"
```

Most of the transformation functions just call basic string manipulation functions (e.g., from `stringr`). For example, the `removeNumbers()` function just removes all numbers

```
text = "04-06-16 Tonight we're going to party like it's 1999!"
tm::removeNumbers(text)
#> [1] "-- Tonight we're going to party like it's !"
stringr::str_replace_all(text, pattern="[:digit:]+", replacement="")
#> [1] "-- Tonight we're going to party like it's !"
```

To apply a transformation to the corpus, you need to use the function `tm_map(<corpus>, <function>)`. For example

```
tmp_corpus = tm_map(corpus, stripWhitespace)
```

will create a new corpus where all *extra* whitespace has been stripped out.

3.1.1 stop words

The `tm` package also gives a list of stop words

```
stopwords("english")
#>   [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"
```

Notice that all of these are lowercase. So to filter these out, we need to *first* transform all letters to lowercase. To remove these words from the corpus use the `removeWords()` function

```
tmp_corpus = tm_map(tmp_corpus, removeWords, stopwords("english"))
```

3.1.2 Multiple transformations

We can link transformations together with the pipe operator (`%>%`)

```
tmp_corpus =
  corpus %>%
    tm_map(stripWhitespace) %>%                      # remove extra whitespaces
    tm_map(content_transformer(str_to_lower)) %>%    # convert to lowercase
    tm_map(removeWords, stopwords("english"))        # remove stop words

as.character(tmp_corpus[[22]])
#> [1] "  simple take: programming  r/python   data analysis   visualization. equally impo
```

Notice that we have reduce the data considerably, but not reduced much information.

Suppose we want to get an idea of what software is popular. In this document, we see “r/python”. We need to be careful how we remove punctuation to ensure we can separate “r” and “python”. If we use tm’s `removePunctuation()` function, then we will have a problem

```
removePunctuation(as.character(tmp_corpus[[22]]))
#> [1] "  simple take programming  rpython   data analysis   visualization equally importa
```

It is also important to recognize the **order of transformation matters**. If all of the stop words are in lowercase, then the text should be converted to lowercase before removing stop words.

3.1.3 Custom Transformations

We can also use custom functions in `tm_map()` as long as the first argument can be a text document. For example, we want to remove punctuation, but add a space between "r/python"

```
str_replace_all(as.character(tmp_corpus[[22]]), "[:punct:]+", " ")
#> [1] "  simple take  programming  r python   data analysis   visualization  equally impo
```

To put this in a form suitable for use in `tm_map()`, we need to use `content_transformer()` like this

```
#- make new function based on str_replace_all()
replace <- content_transformer(stringr::str_replace_all)
```

3.2 Our first attempt at transformations

Here is what I came up with as a first round solution

```
#- additional words to remove
rm_words = c('also', 'areas', 'can', 'etc', 'get', 'just', 'like',
             'lot', 'many', 'may', 'need', 'one', 's', 'set', 't',
             'time', 'us', 'use', 'way', 'well', 'will', 'b', 'e',
             'g', 'less', 'give', 'tell', 'im', 'take', 'coming',
             'say', 'really')


#- make new function based on str_replace_all()
replace <- content_transformer(stringr::str_replace_all)

#- Remember: order matters!
corpus2 = corpus %>%
  tm_map(replace, "'", "") %>%                     # remove apostrophes
  tm_map(replace, "[[:punct:]]+", " ") %>%         # replace (other) punctuation with space
  tm_map(content_transformer(str_to_lower)) %>%    # convert to lowercase
  tm_map(removeWords, stopwords("english")) %>%    # remove stopwords
  tm_map(removeWords, rm_words) %>%                # remove extra words
  tm_map(stripWhitespace)    %>%                   # remove extra whitespaces
  tm_map(removeNumbers)                            # remove *all* numbers
```

3.3 Stemming (and Lemmatization)

We noticed a potential problem when multiple words correspond to the same concept or idea. For example, "analyzing", "analyze", and "analysis" could potentially be grouped together for frequency analysis (note: this could potentially be done after processing, but then we will be forced to deal with much larger data).

Stemming and *Lemmatization* refer to the process of reducing words to a base or root form so multiple words that carry similar meaning/information can be combined. *Stemming* uses letter patterns (think regex) while *lemmatization* finds the part of speech to help guide the stemming. Some more details can be found here http://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html.

The tm package can stem words using Porter's (not me!) stemming algorithm http://snowball.tartarus.org/algorithms/porter/stemmer.html. But this requires functions from the SnowballC package, which must be installed and loaded. Here is an example of how the stemming works

```
library(SnowballC)    # for wordStem() function
filter(total_counts, str_detect(word, pattern="[Aa]naly")) %>%
  mutate(stemmed=wordStem(word))
#> # A tibble: 10 × 3
#>           word     n stemmed
#>          <chr> <int>   <chr>
#> 1    analytics    38  analyt
#> 2    Analytics    31  Analyt
#> 3     analysis    13 analysi
#> 4     analysts     8 analyst
#> 5   analytical     6  analyt
#> 6     analytic     5  analyt
#> 7     Analysis     3 Analysi
#> 8      analyst     2 analyst
#> 9      analyze     1  analyz
#> 10   analyzing     1  analyz
```

Stemming may not great for word cloud, because the stemmed version may not make much sense. One approach is to stem the words, then use one representative word in the word cloud. However, we will not go into this much detail here.

4 Document Term Matrix

The *document term matrix* is a matrix with rows that corresponds to the documents and columns that correspond to words (terms). The function `DocumentTermMatrix(x=<corpus>, control=<list of options>)` generates the matrix.

```
#-- Make Document (sparse) Term Matrix
#   dtm has rows corresponding to documents and columns corresponding to terms
dtm = DocumentTermMatrix(corpus2,
            control=list(wordLengths=c(1,1000)))   # allow one letter words
dim(dtm)
#> [1]   26 1360
```

The help for ?termFreq specifies the control options. Above, we allow words of length 1 so words like "R" and "D3" will be included.

4.1 Modify words

We made everything lowercase, but may want to change these in preparation for making a word cloud. This will illustrate the process of modifying the software related words.

```
#- get words/terms in columns
cnames = colnames(dtm)            # column names (terms)
head(cnames)
#> [1] "able"         "ahead"        "almost"       "analytics"
#> [5] "arena"        "availability"

#-- List of software (in desired case)
software_list = c('R', 'SQL', 'Python', 'Tableau', 'D3', 'MySQL', 'SAS', 'SPSS', 'Excel')
```

Now we need to find the column names that are the lowercase version of the software and replace with the proper case. There are a few ways to do it; here are two.

```r
#- Method 1: loop
 for(i in software_list){
   ind = cnames %in% str_to_lower(i)    # find matches
   cnames[ind] = i                      # replace
 }
```

```r
#- Method 2: str_replace_all with special `pattern=` argument
new_vals = software_list
old_vals = str_to_lower(software_list)
pat = setNames(new_vals, str_c("^",old_vals,"$")) # named vector
   # regex exact match of 'pattern' is "^pattern$"
cnames2 = str_replace_all(cnames, pattern = pat)
```

```r
#- Ensure they are identical
identical(cnames, cnames2)
#> [1] TRUE


#- Reassign column names
colnames(dtm) = cnames               # reassign column names
```

4.2 Word Frequency

It is easy to get the word counts from the document term matrix using the `colSums()` function

```r
term_count =
  data_frame(word = colnames(dtm),
             n=colSums(as.matrix(dtm))) %>%
  arrange(desc(n))
```

The `as.matrix()` is used to convert a *sparse* matrix to a regular matrix.

The `term_count` data frame gives the total number of time a word is used in all the documents. We can also get the number of documents the word appears in by first converting the matrix to a logical (`TRUE` if an element is greater than 0 and `FALSE` otherwise).

```r
distinct_term_count =
  data_frame(word = colnames(dtm),
             n_docs=colSums(as.matrix(dtm)>0)) %>%
  arrange(desc(n_docs))
```

Now we can use `dplyr` tools to find specific words, etc.

```r
a = term_count %>% filter(word %in% software_list)
b = distinct_term_count %>% filter(word %in% software_list)
left_join(a,b, by="word")
#> # A tibble: 8 × 3
#>       word     n n_docs
#>      <chr> <dbl>  <dbl>
#> 1        R    17      9
#> 2      SQL    13      9
#> 3   Python    12      9
#> 4  Tableau     5      4
#> 5      SAS     3      3
#> 6    Excel     2      2
#> 7    MySQL     1      1
#> 8     SPSS     1      1
```

5 Word Clouds

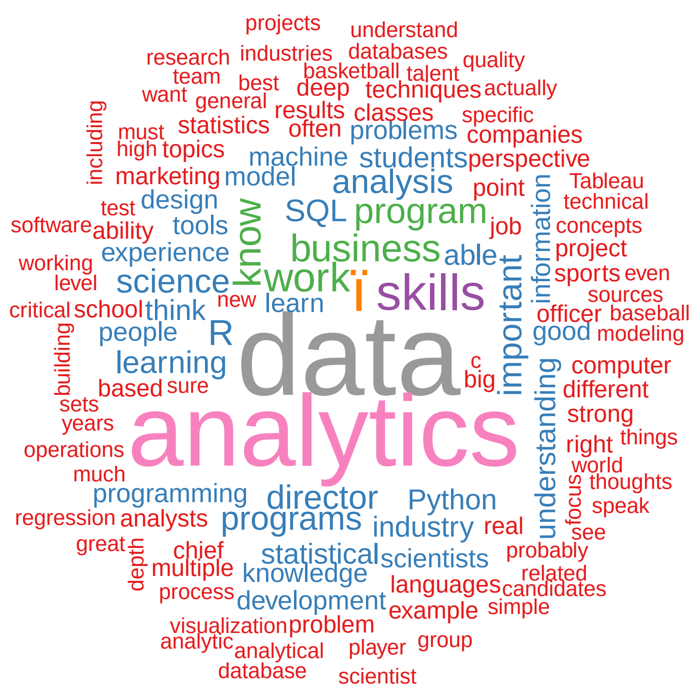
The package `wordcloud` makes word clouds. A word cloud is a graphical representation of text that sizes and colors the words. Size is usually considered to be proportional to the frequency of the word's occurrence, but in general could be related to some other measure of *importance*.

Notice the `wordcloud()` functions requires two vectors (columns of the `term_count` data frame), `words=` and `freq=` and then other options related to the display. In the following, I modify the:

- `scale` of word sizes
- `min.freq=5` to only include words that have freq >= 5
- `random.order=FALSE` to plot words according to frequency
- `colors=brewer.pal(0, "Set1")` to set the color palette. See `brewer.pal.info` for list of palettes.

```
library(wordcloud)   # install.packages("wordcloud")
library(RColorBrewer)
```

```
set.seed(317)            # stochastic layout
wordcloud(term_count$word, term_count$n,
          scale = c(4, .5),
          min.freq = 5,
          random.order = FALSE,
          colors = brewer.pal(9, "Set1")
          )
```




```
set.seed(317)
wordcloud(distinct_term_count$word, distinct_term_count$n_docs,
          scale = c(4, .2),
          min.freq = 4,
          random.order = FALSE,
          colors = brewer.pal(5, "Set1")
          )
```


companies
thoughts
information
related
value
regression
classes
analysis
visualization
large
process
helps
technical
specific
knowledge
model
simple
quality
problems
important
computer
critical
Python
programs
modeling
new
ability
SQL
business
students
program
hope
job
skills
director
real
results
officer
best
focus
analytics
tools
good
point
much
big
design
want
learn
R
data
industry
machine
know
team
people
work
ï
think
learning
able
world
years
level
ask
statistical
science
topics
see
often
high
school
sets
group
understanding
different
understand
programming
statistics
general
based
strong
marketing
probably
concepts
example