R Tutorial 3

William Bell

2019-03-10

This will be our final tutorial, and we will cover the tools specifically necessary to process and investigate text data. This has three parts, text manipulation, text-specific investigative approaches, and textual data visualization.

To start off, I will use the #dog dataset that we've investigated previously.

```
dfdogs <- read.csv("../dfdogs.csv", stringsAsFactors = F)
```

A valuable first step is to look at all of the columns, and see which ones might be worthwhile to consider:

```
colnames(dfdogs)
```

```
##  [1] "X"             "text"          "favorited"     "favoriteCount"
##  [5] "replyToSN"     "created"       "truncated"     "replyToSID"
##  [9] "id"            "replyToUID"    "statusSource"  "screenName"
## [13] "retweetCount"  "isRetweet"     "retweeted"     "longitude"
## [17] "latitude"
```

The text of the tweet is going to be the main focus for today's tutorial, but you could just as easily determine the proportion of tweets that are retweets from the "isRetweet" column, the number of times the average dog post is retweeted or favourited from the "retweetCount" and "favorited" columns, or map the tweets using the latitude/longitude information.

However today we'll focus on the text of the tweets:

```
specialPrint(dfdogs$text[1:5])
```

```
## [1] "RT @LovUniverse: Please #help #donate #support #GermanShepherd #Animals #Dogs ht..."
## [2] "#Odin stayed with his family's 8 goats through the #wildfires that raged around ..."
## [3] "RT @nowthatsbully: Who’s side are you on? #dogs #pets https://t.co/Y743YWznCd..."
## [4] "RT @Cory__1077: I just wanted to say a big THANK YOU to everyone today for all t..."
## [5] "RT @nowthatsbully: Who’s side are you on? #dogs #pets https://t.co/Y743YWznCd..."
```

One possibly worthwhile project could be finding every instance of a character string. Say, how many include the word "pet". There are two ways to go about this, depending on how you want the information to be presented to you. If you want the row number of every tweet with "pet" in it, you can use `grep` (**g**lobally search **r**egular **e**xpression **p**rint):

```
indexPets <- grep("pet", dfdogs$text)
indexPets
```

```
##  [1]   3   5  10  14  15  25  26  27  33  47  51  53  56  62  75  83  86
## [18]  91 100
```

The other way makes a logical vector, and is called `grepl`:

```
logicalPets <- grepl("pet", dfdogs$text)
logicalPets
```

```
##   [1] FALSE FALSE  TRUE FALSE  TRUE FALSE FALSE FALSE FALSE  TRUE FALSE
##  [12] FALSE FALSE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [23] FALSE FALSE  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE  TRUE
##  [34] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

```
##  [45] FALSE FALSE  TRUE FALSE FALSE FALSE  TRUE FALSE  TRUE FALSE FALSE
##  [56]  TRUE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE
##  [67] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE
##  [78] FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE  TRUE FALSE FALSE
##  [89] FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [100]  TRUE
```

Which is preferable depends on the context and what you're used to. For instance, here's how you'd do a few basic operations given these tools:

```
## Get all of the ones involving pets:

pets <- dfdogs$text[indexPets]
pets <- dfdogs$text[logicalPets]

## Get all of the ones NOT involving pets:

notpets <- dfdogs$text[setdiff(1:nrow(dfdogs), indexPets)]
notpets <- dfdogs$text[!logicalPets]

## How many have pets?

howmany <- length(indexPets)
howmany <- sum(logicalPets)

## What proportion have pets?

whatproportion <- length(indexPets)/nrow(dfdogs)
whatproportion <- mean(logicalPets)
```

I think that `grepl` is generally simpler, but that's really up to you.

One thing that you might want to do is clear up punctuation. Afterall, you don't want "not" and "not." to be different words if we break up words by spaces. For these, we can use `gsub` which searches for a pattern and replaces it with whatevery you tell it to. For instance, you can replace "didn't" with "did not" as follow:

```
removeNT <- gsub("didn't", "did not", dfdogs$text)
```

In order to remove all punctuation, I will give you a function that does the trick, but don't worry about understanding it necessarily:

```
noGrammar <- function(data) {gsub('[[:punct:] ]+','', data, perl = T)}

removeGrammar <- noGrammar(dfdogs$text)
specialPrint(removeGrammar[1:5])
```

```
## [1] "RTLovUniversePleasehelpdonatesupportGermanShepherdAnimalsDogshttpstcokO96UEYkpp..."
## [2] "Odinstayedwithhisfamilys8goatsthroughthewildfiresthatragedaroundthefamilyshousei..."
## [3] "RTnowthatsbullyWho’ssideareyouondogspetshttpstcoY743YWznCd..."
## [4] "RTCory1077IjustwantedtosayabigTHANKYOUtoeveryonetodayforallthekindnessandsupport..."
## [5] "RTnowthatsbullyWho’ssideareyouondogspetshttpstcoY743YWznCd..."
```

You might want to find the number of characters in a particular string alternatively:

```
nchar(dfdogs$text[1])
```

```
## [1] 101
```

Who knows, maybe people who tweet about pets hit the character limit more often than people who just talk about dogs in general.

Or you might want to take out a certain group of characters in each string:

```
substr(dfdogs$text, 1, 2)[1:10]
```

```
##  [1] "RT" "#O" "RT" "RT" "RT" "RT" "#d" "RT" "RT" "RT"
```

Finally I will suggest one other task that could be useful for you to perform, and that is to split up strings according to some rule:

```
words <- strsplit(dfdogs$text, " ")
```

In this case I took all of our tweets and broke them up at every space, this has the effect of leaving just the words for each tweet:

```
words[1:3]
```

```
## [[1]]
##  [1] "RT"                           "@LovUniverse:"
##  [3] "Please"                       "#help"
##  [5] "#donate"                      "#support"
##  [7] "#GermanShepherd"              "#Animals"
##  [9] "#Dogs"                        "https://t.co/kO96UEYkpp"
##
## [[2]]
##  [1] "#Odin"                        "stayed"
##  [3] "with"                         "his"
##  [5] "family's"                     "8"
##  [7] "goats"                        "through"
##  [9] "the"                          "#wildfires"
## [11] "that"                         "raged"
## [13] "around"                       "the"
## [15] "family's"                     "house"
## [17] "in"                           "#California"
## [19] "i..."                           "https://t.co/vvDLOXeZwq"
##
## [[3]]
##  [1] "RT"                           "@nowthatsbully:"
##  [3] "Who’s"                        "side"
##  [5] "are"                          "you"
##  [7] "on?"                          "#dogs"
##  [9] "#pets"                        "https://t.co/Y743YWznCd"
```

This will be essential for the next set of tasks. I am going to make a function that takes in a set of tweets, and outputs all of the words used in those tweets, with a number identifying the tweet they were used in beside them. You don’t have to follow along while I’m making this function, but you should know how to use it.

```
tweetsToWords <- function(tweets) {
  wordsTweet <- strsplit(tweets, split = " ")
  wordsEach <- sapply(wordsTweet, length)
  wordStart <- c(0, cumsum(wordsEach))
  totalWords <- sum(wordsEach)
  totalTweets <- length(wordsEach)
  key <- rep(NA, totalWords)
  words <- rep(NA, totalWords)
  for (j in seq_len(totalTweets)) {
    key[(wordStart[j]+1):wordStart[j+1]] <- j
```

```
    words[(wordStart[j]+1):wordStart[j+1]] <- wordsTweet[[j]]
  }
  words <- noGrammar(words)
  wordsdf <- data.frame(tweets = key, word = words, stringsAsFactors = F)
  wordsdf
}

psych::headTail(tweetsToWords(dfdogs$text))
```

```
##      tweets         word
## 1         1           RT
## 2         1  LovUniverse
## 3         1       Please
## 4         1         help
## ...     ...         <NA>
## 1849    100           to
## 1850    100     updating
## 1851    100 shareholders
## 1852    100          v...
```

We see that the words are all laid out, and they're given a number to represent which tweet they were taken from. From here, there are many things we can do, the most basic is we can look at the frequency of words across every tweet in our dataset using the `table` function (also works on things that aren't words!):

```
dogWords <- tweetsToWords(dfdogs$text)

sort(table(dogWords$word), decreasing = T)[1:10]
```

```
##
##        RT dogs  the   to    a   is   of   in  for
##  172   75   59   44   25   22   22   20   19   18
```

Ignoring the empty ones, we see that RT is up there, as is dogs, the, to, a, is, of, in… Which isn't surprising but probably not what we're looking for. What we're looking for is probably certain *poignant* words, words that mean something to somebody. For this, we now turn to the field of **sentiment analysis**.

Problems

1. Take the first word from each tweet with `strsplit` and a loop.
2. How many tweets use the word "love"?
3. How many tweets mention German Shepherds? Consider multiple different ways of writing German Shepherd (e.g. "German Shepherd", "#GermanShepherd", "german shepherd")? Look at the `ignore.case` argument of the `grep`/`grepl` functions

Sentiment Analysis

For sentiment analysis work, the main focus is on tagging as many of the words in your dataset as possible to determine their sentimental significance. For instance, you can ask what proportion of the words shared under a given hashtag are negative or positive and compare that to other hashtags. For this kind of work, we need a lot of tagged words, as it happens the tidytext package has exactly what we need for this:

```
if (!("tidytext" %in% row.names(installed.packages()))) {
    install.packages("tidytext")
```

```r
}

library(tidytext)

psych::headTail(sentiments)
```

```
##          word   sentiment  lexicon score
## 1      abacus       trust      nrc  <NA>
## 2     abandon        fear      nrc  <NA>
## 3     abandon    negative      nrc  <NA>
## 4     abandon     sadness      nrc  <NA>
## 5        <NA>        <NA>     <NA>   ...
## 6      theses superfluous loughran  <NA>
## 7  ubiquitous superfluous loughran  <NA>
## 8 wheresoever superfluous loughran  <NA>
## 9      whilst superfluous loughran  <NA>
```

```r
nrow(unique(sentiments[ , 1]))
```

```
## [1] 13922
```

```r
nrow(sentiments)
```

```
## [1] 27314
```

There are 14,000 (!!) unique words with 27,000 tags in total in this dataset. The words tagged are in the first column, and the sentiment they're supposed to express are in the second column.

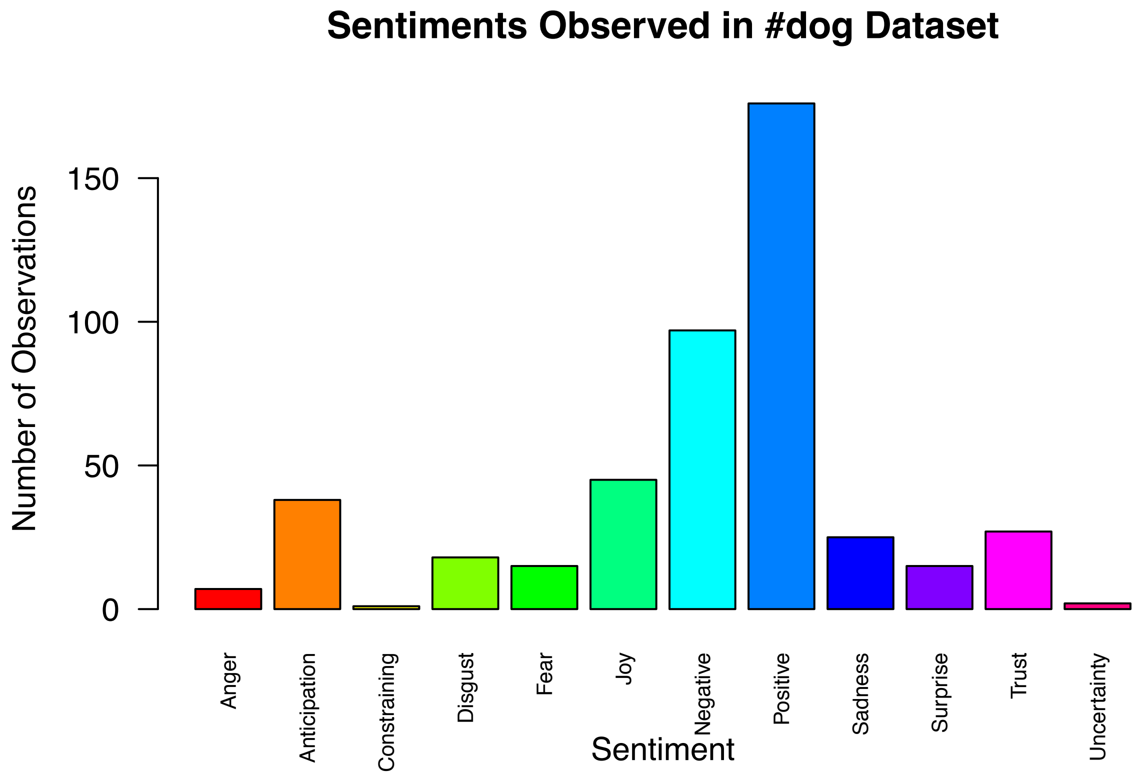
So now we have two datasets, one is a list of words and the tweets they're associated with, and one is a list of words and the sentiments they're associated with. This is a perfect case for the joins we learned last time, and in particular `inner_join`:

```r
suppressPackageStartupMessages(library(dplyr))

dogWords$word <- stringr::str_to_lower(dogWords$word)

sentimentalDogWords <- inner_join(dogWords, sentiments, by = "word")

psych::headTail(sentimentalDogWords)
```

```
##     tweets     word sentiment  lexicon score
## 1        1   please      <NA>    AFINN     1
## 2        1     help      <NA>    AFINN     2
## 3        1  support  positive     bing  <NA>
## 4        1  support      <NA>    AFINN     2
## ...    ...     <NA>      <NA>     <NA>   ...
## 568    100 exciting  positive loughran  <NA>
## 569    100    works  positive     bing  <NA>
## 570    100      pet  negative      nrc  <NA>
## 571    100  forward  positive      nrc  <NA>
```

Now we can see which sentiments are expressed most commonly:

```r
barplot(table(stringr::str_to_title(sentimentalDogWords$sentiment)),
        las = 2, col = rainbow(12), main = "Sentiments Observed in #dog Dataset",
        xlab = "Sentiment", ylab = "Number of Observations", cex.names = 0.65)
```



So we can see that over a hundred of the words used are tagged positive, about 40 negative (must be cat people), then anticipation ("Rex is **looking/waiting** for his forevery home!"), joy, and trust follow. Overall, you're probably saying pretty positive things if you're using the #dog hashtag, which might be what you expected.

However hopefully you see how you could apply this to a more serious dataset. It might be worthwhile to track whether the proportion of positive coverage for instance produces a greater or smaller twitter response (in terms of total number of tweets). Since this could change what methods we should use to publicize disasters. It might be good to know if the sentiments relating to a natural disaster change over time (even in real time - a turn negative could indicate that a location is experiencing trouble before that information reaches the news).

You can imagine other things we can do with this data, we can now look at which words occur most frequently among the words with sentimental value:

```
sort(table(unique(sentimentalDogWords[ , 1:2])$word), decreasing = T)[1:10]
```

```
##
##     love     lost      pet exciting  forward    works    great   please
##       15        9        8        6        6        6        5        5
##     sick     like
##        5        4
```

Conclusion

I introduced a couple miccelaneous useful functions along the way, such as `unique`, `table`, `sort`, `stringr::str_to_lower` (turns the characters lower case), and `stringr::str_to_title` (turns the characters to capitalization like in a title). However the main focus has been on string manipulation functions, especially `grep`, `grepl`, `gsub`, `substr`, and `strsplit`. We've made the most use out of `grepl`, `gsub`, and `strsplit` but they're all useful. I made a special function which would do some of the hard work for you (`tweetsToWords`), which if you plan to try looking at the text of any tweets, you should use. Then we looked at the basics of sentiment analysis, and some of the analysis you can do with sentiment information.