tm Polarity Scoring - Exercises

April 28, 2023

Contents

1 Exercise: `polarity` scoring **2**

2 Exercise: `qdap`'s lexicon **3**

3 Exercise: examine and use `qdap`'s lexicon **6**

4 Exercise: amplification and negation words **9**

Load data for the exercises:

```
load("~/Documents/GitHub/tm/data/.RData")
ls()
library(qdap)
search()
```

```
 [1] "api_key"                "ask_chatgpt"            "chardonnay_corpus"
 [4] "chardonnay_df"          "chardonnay_src"         "chardonnay_vec"
 [7] "clean_chardonnay"       "clean_chardonnay_corpus" "clean_coffee"
[10] "clean_coffee_corpus"    "coffee_corpus"          "coffee_df"
[13] "coffee_src"             "coffee_vec"             "load_packages"
[16] "m"                      "sms_corpus"             "sms_corpus_clean"
[19] "sms_dtm"                "sms_dtm2"               "sms_raw"
 [1] ".GlobalEnv"              "package:qdap"             "package:RColorBrewer"
 [4] "package:qdapTools"       "package:qdapRegex"        "package:qdapDictionaries"
 [7] "ESSR"                    "package:stats"            "package:graphics"
[10] "package:grDevices"       "package:utils"            "package:datasets"
[13] "package:stringr"         "package:httr"             "package:methods"
[16] "Autoloads"               "package:base"
```

1 Exercise: `polarity` scoring

- `polarity` scans the text to identify words in the lexicon. It then creates a *cluster* around an identified *subjectivity word*. Within the cluster *valence shifters* adjust the score.
- Valence shifters are words that amplify or negate the emotional intent of the subjectivity word. For example, "well known" is positive while "not well known" is negative. Here "not" is a negating term and reverses the emotional intent of "well known." In contrast, "very well known" employs an *amplifier* increasing the positive intent.
- The `polarity` function then calculates a score using subjectivity terms, valence shifters and the total number of words in the passage. This exercise demonstrates a simple polarity calculation.
- Calculate the `polarity` of the string `positive` in a new object called `pos_score`, then print it:

```
library(qdap)
positive <- "DataCamp courses are good for learning"
# Calculate polarity of statement
pos_score <-polarity(positive)
pos_score
```

```
  all total.sentences total.words ave.polarity sd.polarity stan.mean.polarity
1 all               1           6        0.408          NA                 NA
```

- Manually perform the same polarity calculation:
  1. Get a word count object `pos_counts` by calling `counts` on the polarity object `pos_score`, then print it:

     ```
     pos_counts <- counts(pos_score)
     pos_counts
     ```

     ```
       all wc polarity pos.words neg.words                                text.var
     1 all  6    0.408      good         - DataCamp courses are good for learning
     ```

  2. All the identified subjectivity words are part of count object's list. Specifically, positive words are in the `$pos.words` element vector of `pos_counts`. Print the structure of `pos_counts`:

```
str(pos_counts)
```

```
Classes 'polarity_count' and 'data.frame': 1 obs. of  6 variables:
 $ all      : chr "all"
 $ wc       : int 6
 $ polarity : num 0.408
 $ pos.words:List of 1
  ..$ : chr "good"
 $ neg.words:List of 1
  ..$ : chr "-"
 $ text.var : chr "DataCamp courses are good for learning"
 - attr(*, "type")= chr "polarity_count"
 - attr(*, "digits")= num 3
```

3. Find the number of positive words by calling `length` on the first member of the `$pos_words` element of `pos_counts` and store it in `n_good`:

```
length(pos_counts[[1]]) -> n_good
```

4. Capture the total number of words and assign it to `n_words`. This value is stored in `pos_count` as the `wc` (word count) element:

```
pos_counts$wc -> n_words
```

5. De-construct the `polarity` calculation by dividing `n_good` by `sqrt` of `n_words` and save the result as `pos_pol`. Compare `pos_pol` to `pos_score` calculated with `polarity` earlier:

```
n_good/sqrt(n_words) -> pos_pol
identical(pos_pol,pos_score$all[[3]])
```

```
[1] TRUE
```

2 Exercise: qdap's lexicon

- Of course just positive and negative words aren't enough. In this exercise you will learn about valence shifters which tell you about the author's emotional intent. Previously you applied `polarity()` to text without valence shifters. In this example you will see amplification and negation words in action.
- Recall that an amplifying word adds 0.8 to a positive word in polarity() so the positive score becomes 1.8. For negative words 0.8 is subtracted

so the total becomes -1.8. Then the score is divided by the square root of the total number of words.

- Consider the following example from Frank Sinatra:

  ```
  "It was a very good year"
  ```

  "Good" equals 1 and "very" adds another 0.8. So, 1.8/sqrt(6) results in 0.73 polarity.

- A negating word such as "not" will inverse the subjectivity score. Consider the following example from Bobby McFerrin:

  ```
  "Don't worry Be Happy"
  ```

  ```
  [1] "Don't worry Be Happy"
  ```

  "worry is now 1 due to the negation "don't." Adding the "happy", +1, equals 2. With 4 total words, 2 / sqrt(4) equals a polarity score of 1.

  Exercise:

  1. Load the `conversation` data frame:

     ```
     conversation <- data.frame( "student"=c("Martijn","Nick","Nicole"),
       "text"=c("This restaurant is never bad", "The lunch was very good",
       "It was awful I got food poisoning and was extremely ill"))
     str(conversation)
     ```

     ```
     'data.frame': 3 obs. of  2 variables:
      $ student: chr  "Martijn" "Nick" "Nicole"
      $ text   : chr  "This restaurant is never bad" "The lunch was very good" "It
     ```

  2. Examine the `conversation` data frame.

     ```
     conversation
     ```

     ```
       student                                                  text
     1 Martijn                              This restaurant is never bad
     2    Nick                                   The lunch was very good
     3  Nicole It was awful I got food poisoning and was extremely ill
     ```

  3. What context cluster category is "never"?

Answer: a valence shifter - it affects the emotional content and shifts the polarity of "bad" to positive.

4. Apply `polarity()` to the text column of conversation to calculate the polarity for the entire conversation:

```
polarity(conversation$text)
```

```
  all total.sentences total.words ave.polarity sd.polarity stan.mean.polarity
1 all               3          21        0.317       0.565              0.561
```

5. Calculate the polarity scores for the `text` by `student` using the `grouping.var` argument, and assign the result to `student_pol`:

```
polarity(text.var=conversation$text,
         grouping.var=conversation$student) -> student_pol
```

6. To see the `student` level results, use `scores()` on `student_pol`:

```
scores(student_pol)
```

```
  student total.sentences total.words ave.polarity sd.polarity stan.mean.pola
1 Martijn               1           5        0.447          NA
2    Nick               1           5        0.805          NA
3  Nicole               1          11       -0.302          NA
```

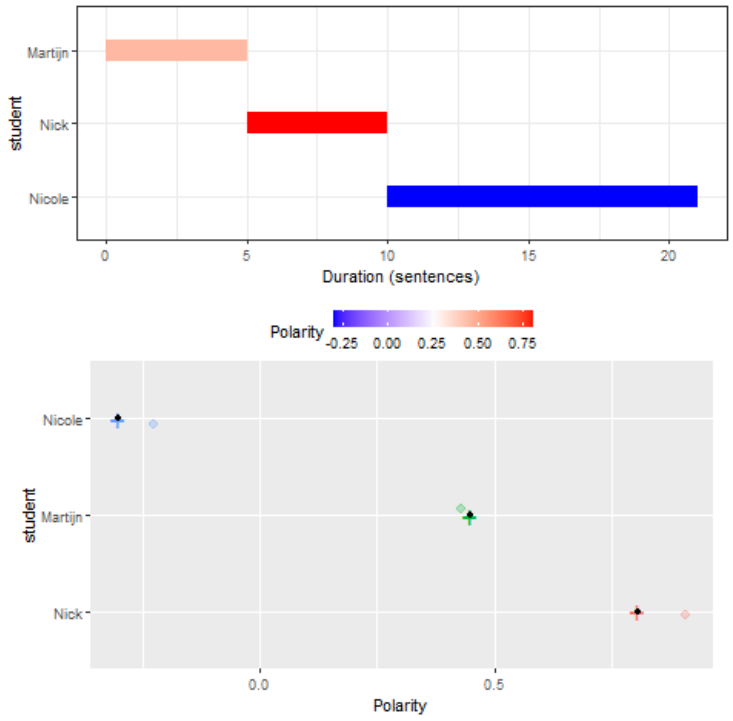
7. The `counts()` function applied to `student_pol` will print the sentence level polarity for the entire data frame along with lexicon words identified:

```
counts(student_pol)
```

```
  student wc polarity pos.words neg.words
1 Martijn  5    0.447         -       bad                            This res
2    Nick  5    0.805      good         -                                The
3  Nicole 11   -0.302         -     awful It was awful I got food poisoning a
```

8. The polarity object, `student_pol`, can be plotted with `plot()`:

```
plot(student_pol)
```



3 Exercise: examine and use `qdap`'s lexicon

- Even with Zipf's law in action, you will still need to adjust lexicons to fit the text source (for example twitter versus legal documents) or the author's demographics (teenager versus the elderly). This exercise demonstrates the explicit components of `polarity()` so you can change it if needed.
- In Trey Songz "Lol :)" song there is a lyric "LOL smiley face, LOL smiley face." In the basic `polarity()` function, "LOL" is not defined as positive. However, "LOL" stands for "Laugh Out Loud" and should be positive. As a result, you should adjust the lexicon to fit the text's context which includes pop-culture slang. If your analysis contains text from a specific channel (Twitter's "LOL"), location (Boston's "Wicked Good"), or age group (teenagers' "sick") you will likely have to adjust the lexicon.
- In the first exercise, you are examining the existing word data frame objects so you can change them in the following exercise.

1. As a sample text, here are two excerpts from Beyoncé's "Crazy in Love" lyrics for the exercise - run the code:

```
text <- data.frame(
  "speaker"=c("beyonce","jay_z"),
  "words"=c("I know I dont understand Just how your love can do what no one else c
            "They cant figure him out they like hey, is he insane"))
str(text)

'data.frame': 2 obs. of  2 variables:
 $ speaker: chr  "beyonce" "jay_z"
 $ words  : chr  "I know I dont understand Just how your love can do what no one e
```

2. Print `qdapDictionaries::key.pol` to see a portion of the subjectivity words and values:

```
qdapDictionaries::key.pol  ## if qdap is loaded, key.pol is sufficient

       x  y
   1:     a plus  1
   2:   abnormal -1
   3:    abolish -1
   4: abominable -1
   5: abominably -1
  ---
6775:  zealously -1
6776:     zenith  1
6777:       zest  1
6778:      zippy  1
6779:     zombie -1
```

3. Examine the predefined `negation.words` to print all the negating terms:

```
negation.words

 [1] "ain't"     "aren't"    "can't"     "couldn't"  "didn't"    "doesn't"
 [7] "don't"     "hasn't"    "isn't"     "mightn't"  "mustn't"   "neither"
[13] "never"     "no"        "nobody"    "nor"       "not"       "shan't"
[19] "shouldn't" "wasn't"    "weren't"   "won't"     "wouldn't"
```

4. Print the amplifiers in `amplification.words` to see the words that add values to the lexicon:

```
amplification.words
```

```
 [1] "acute"         "acutely"       "certain"       "certainly"     "colossal"
 [6] "colossally"    "deep"          "deeply"        "definite"      "definitely"
[11] "enormous"      "enormously"    "extreme"       "extremely"     "great"
[16] "greatly"       "heavily"       "heavy"         "high"          "highly"
[21] "huge"          "hugely"        "immense"       "immensely"     "incalculable
[26] "incalculably"  "massive"       "massively"     "more"          "particular"
[31] "particularly"  "purpose"       "purposely"     "quite"         "real"
[36] "really"        "serious"       "seriously"     "severe"        "severely"
[41] "significant"   "significantly" "sure"          "surely"        "true"
[46] "truly"         "vast"          "vastly"        "very"
```

5. Check the `deamplification.words` that reduce the lexicon values:

```
library(qdap)
deamplification.words
```

```
 [1] "barely"       "faintly"      "few"          "hardly"       "little"
 [6] "only"         "rarely"       "seldom"       "slightly"     "sparsely"
[11] "sporadically" "very few"     "very little"
```

6. Now, calculate the `polarity` of `text` as follows and save it in `text_pol`:

   (a) Set `text.var` to `text$words`.

   (b) Set `grouping.var` to `text$speaker`.

   (c) Set `polarity.frame` to `key.pol`.

   (d) Set `negators` to `negation.words`.

   (e) Set `amplifiers` to `amplification.words`.

   (f) Set `deamplifiers to deamplification.words`.

```
polarity(text.var=text$words,
         grouping.var=text$speaker,
         polarity.frame=key.pol,
         negators=negation.words,
         amplifiers=amplification.words,
         deamplifiers=deamplification.words) -> text_pol
text_pol
```

```
  speaker total.sentences total.words ave.polarity sd.polarity stan.mean.polarity
1 beyonce               1          16         0.25          NA                 NA
2   jay_z               1          11         0.00          NA                 NA
```

7. Print the positive and negative words alongside the `text` with the `all` element of `text_pol`:

```
text_pol$all
```

```
  speaker wc polarity pos.words neg.words
1 beyonce 16     0.25      love         -
2   jay_z 11     0.00      like    insane
                                                                    text.var
1 I know I dont understand Just how your love can do what no one else can
2                    They cant figure him out they like hey, is he insane
```

8. Why is the polarity of Beyonce's lyrics 0.25, and why is the polarity of Jay Z's lyrics 0?

   Answer:

   - Beyonce: (love) 1/sqrt(16)=1/4=0.25
   - Jay Z: +1 (like) -1 (insane) = 0

4 Exercise: amplification and negation words

- Here you will adjust the negative words to account for the specific text. You will then compare the basic and custom `polarity()` scores.
- A popular song from Twenty One Pilots is called "Stressed Out" (2015). If you scan the song lyrics, you will observe the song is about youthful nostalgia. Overall, most people would say the polarity is negative. Repeatedly the lyrics mention stress, fears and pretending.
- Let's compare the song lyrics using the default subjectivity lexicon and also a custom one.
- To start, you need to verify the `key.pol` subjectivity lexicon does not already have the term you want to add. One way to check is with `grep`. The pattern matching `grep()` function returns the row containing characters that match a search `pattern`. Here is an example where the column `col` of `df` is searched for "search$_{\text{pattern}}$":

```
idx <- grep(pattern="search_pattern", x=df$col)
```

- The vector `idx` can now be used to return all elements of `df` that match the pattern as `df[idx, ]`.
- After verifying the slang or new word is not already in the `key.pol` lexicon you need to add it.

---

1. Add the lyrics as a single string from the file `stressed_out.txt` and store it in the vector `stressed_out`, then replace `\\` by `\` with `gsub` and print the lyrics:

```
stressed_out <- readLines("https://bit.ly/stressed_out_txt")
gsub("\\\\n","\n",stressed_out) -> stressed_out
stressed_out
```

```
[1] "I wish I found some better sounds no one's ever heard\nI wish I had a better
```

2. Compute the default `polarity` score of `stressed_out`:

```
polarity(stressed_out)
```

```
  all total.sentences total.words ave.polarity sd.polarity stan.mean.polarity
1 all               1         526       -0.253          NA                 NA
```

3. Bonus question: can you show just the value for the polarity? Tip: `polarity(stressed_out)` is a `list` and "polarity" is a member of the `$all` element of that list (you can check that with `str`):

```
polarity(stressed_out)$all[["polarity"]]
```

```
[1] -0.252892
```

4. Check `key.pol` for any words containing "stress":
   (a) use `grep` to index the data frame by searching in the `x` column
   (b) save the result in `rowindex`

```
idx <- grep(pattern="stress",x=key.pol$x)
key.pol[idx,]
```

```
       x  y
1:      distress -1
2:    distressed -1
3:   distressing -1
4: distressingly -1
5:      mistress -1
6:        stress -1
7:      stresses -1
8:     stressful -1
9:   stressfully -1
```

5. Construct a new polarity lexicon `custom_pol` using `sentiment_frame`. This function creates a sentiment lookup table for use with the `polarity.frame` argument of `polarity` (i.e. the lexicon) - check the function's arguments:

```
args(sentiment_frame)
```

```
function (positives, negatives, pos.weights = 1, neg.weights = -1)
NULL
```

6. Pass `positive.words` as `positives` argument to the function `sentiment_frame`, and for the second argument concatenate (with `c`) `negative_words` and the words "stressed" and "turn back". Save the result in `custom_pol`

```
sentiment_frame(
  positive.words,
  c(negative.words,"stressed","turn back")) -> custom_pol
```

7. Compute the `polarity` using the `custom_pol` lexicon as `polarity.frame`:

```
polarity(stressed_out, polarity.frame = custom_pol)$all[c("polarity")]
```

```
   polarity
1 -0.819719
```

8. You should see that the modified lexicon leads to a more realistic sentiment scoring than the standard lexicon.