Text analysis workshop: Basic sentiment analysis

Casey O'Hara

Overview

Sentiment analysis is a fairly basic way to get a sense of the mood of a piece of text. In an eco-data-science sense, we can use sentiment analysis to understand perceptions of topics in environmental policy.

A good example is "Public Perceptions of Aquaculture: Evaluating Spatiotemporal Patterns of Sentiment around the World" by local celebrities Halley Froehlich, Becca Gentry, and Ben Halpern, in which they examine public pereptions of aquaculture by performing sentiment analyses on newspaper headlines from around the globe and government-solicited public comments on aquaculture policy and development. This paper is included in the 'pdfs' folder on Github, or available here: http://journals.plos.org/plosone/article?id=10.1371/journal.pone.0169281

Another popular use of sentiment analysis is to determine the mood of Twitter comments. One excellent example is an examination of Trump tweets, which noted that tweets from an iPhone and an Android phone were markedly different in tone; the thought was that the Android account (with generally far more positive tweets) was run by a staffer while the iPhone (with generally more negative tweets) was Trump's personal tweets. See: http://varianceexplained.org/r/trump-tweets/

Sentiment analysis of text from a PDF

We'll try twitter in the next exercise; but first, let's just look at how to do a sentiment analysis on a PDF since we now know how to get the text from PDFs. We will need the `tidyverse`, `stringr`, and `pdftools` packages, but also the `tidytext` package which makes basic sentiment analysis pretty easy.

```
library(tidyverse)
library(stringr)

library(pdftools)
library(tidytext)
```

There's a great tutorial on using `tidytext` to do a sentiment analysis on the Lord of the Rings (https://www.r-bloggers.com/sentiment-analysis-of-the-lord-of-the-rings-with-tidytext/), so I adapted that for this workshop, and will use A Game of Thrones as an example.

I've also included PDFs I found online for The Hobbit, Harry Potter and the Sorcerer's Stone, and some Kurt Vonnegut short stories, in case you'd like to try one of those instead.

Loading and cleaning the text

We will use some tricks to load the text and clean it up a bit for easier processing.

- Get the text using `pdftools::pdf_text` and turn it into a dataframe, including page numbers.
- Filter to the page numbers of the actual text (no table of contents, preface, etc)
- get chapter info and attach that to each page. We'll analyze the overall sentiment of each chapter separately.
- Clean the text by converting all to lower case, then splitting each page into individual lines.

```
got <- pdf_text('pdfs/got.pdf')
got_df <- data.frame(text = got) %>%
  mutate(text = tolower(text),
         page = 1:n())

main_text <- got_df %>%
  filter(page %in% 6:731) %>%
  mutate(text = str_split(text, '\n')) %>%
  unnest(text)

ch_list <- main_text %>%
  group_by(page) %>%
  mutate(ch = ifelse(str_detect(first(text), 'previous.+table of contents.+next'),
                     nth(text, 2), NA)) %>%
  filter(text == first(text)) %>%
  filter(!is.na(ch)) %>%
  ungroup() %>%
  mutate(ch_num = 0:(n()-1)) %>%
  select(ch, ch_num, page)

text_w_ch <- main_text %>%
  select(page, text) %>%
  left_join(ch_list, by = 'page') %>%
  fill(ch, .direction = 'down') %>%
  fill(ch_num, .direction = 'down')
```

Sentiment analysis basics

The way a basic sentiment analysis works is to assign values to certain words - e.g. "positive" vs "negative" or more specific moods. The `tidytext` package has a few common sentiment libraries built in.

There is also a `stop_words` dataframe of common words (e.g. "the", "a", "an", etc) that don't convey sentiment but just take up space in our analysis. We can (but don't have to) ditch these to clean up our analysis.

```
head(stop_words) ### common words that won't impact analysis
```

```
## # A tibble: 6 x 2
##        word lexicon
##       <chr>   <chr>
## 1         a   SMART
## 2       a's   SMART
## 3      able   SMART
## 4     about   SMART
## 5     above   SMART
## 6 according   SMART
```

The `tidytext::get_sentiments()` function returns a dataframe of words and their associated sentiments. There are four built-in as of now; you could also adapt or create your own sentiment library (and we'll see why soon).

```
sentiments_b <- get_sentiments('bing')
# sentiments_a <- get_sentiments('afinn')
# sentiments_n <- get_sentiments('nrc')
```

```
# sentiments_l <- get_sentiments('loughran')
```

The Bing sentiment library is a basic one (just "positive"/"negative") so we'll go with that, but consider the others as well in terms of advantages/disadvantages for your particular needs.

Perform the sentiment analysis

The steps here:

- Break the entire text into individual words
- Eliminate stop words (using `dplyr::anti_join` is handy here)
- Attach sentiment scores by word (using `dplyr::inner_join` is handy here)

```
text_words <- text_w_ch %>%
  tidytext::unnest_tokens(output = word, input = text, token = 'words') %>%
  anti_join(stop_words, by = 'word') %>%
  inner_join(sentiments_b, by = 'word')
```

For each chapter, add up the positives and negatives (using `dplyr::count()` is handy here), spread them out into columns and take the difference (number of positives - number of negatives).

Subtracting the overall mean difference from each chapter's difference will center the scores, essentially to account for a writer's general tone. A Game of Thrones is overall pretty dark, so the average sentiment will be skewed negative...

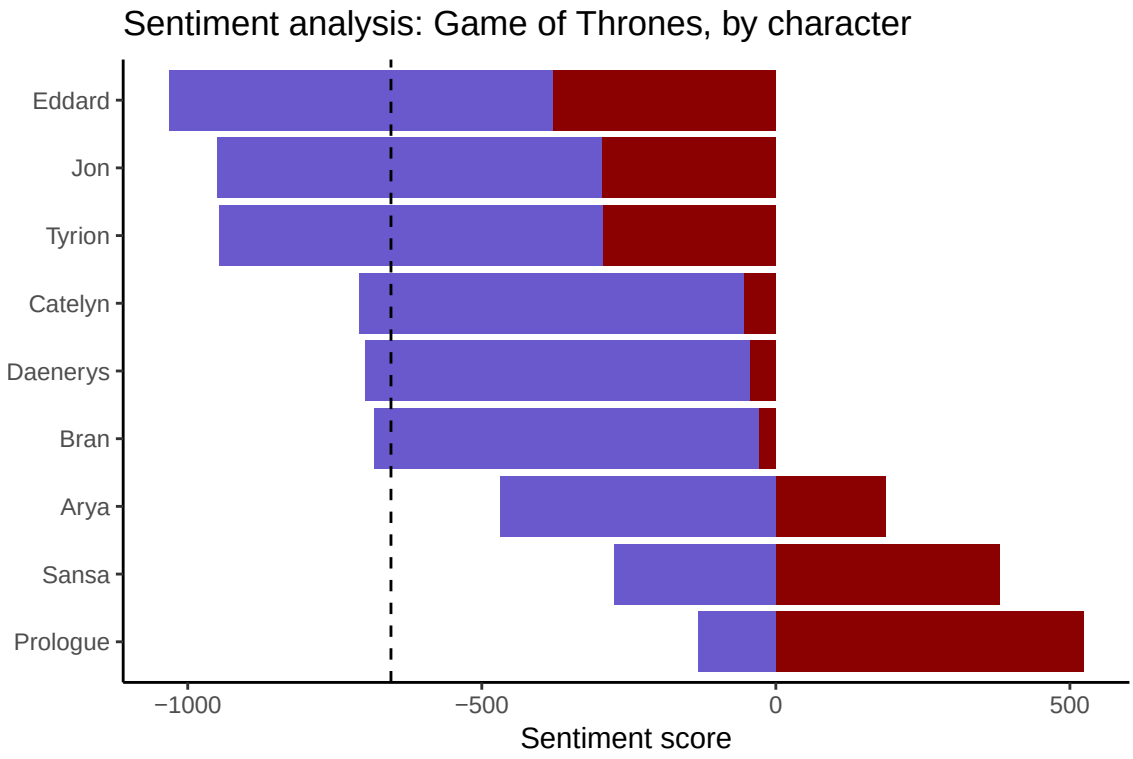
We can also add in a line to order the characters from worst outcome to best outcome. To do so, we turn it into a factor with levels based on the *reverse* order of the raw score (or offset score).

```
text_scores <- text_words %>%
  mutate(ch = str_trim(ch) %>% tools::toTitleCase()) %>%
  count(sentiment, ch) %>%
  spread(sentiment, n) %>%
  mutate(raw_score    = positive - negative,
         offset       = mean(positive - negative),
         offset_score = (positive - negative) - offset) %>%
  mutate(ch = factor(ch, levels = ch[order(desc(raw_score))]))
```

Time permitting: plot it!

Spoiler alerts!

```
ggplot(text_scores, aes(x = ch)) +
  theme_classic() +
  geom_bar(aes(y = raw_score), stat = 'identity', fill = 'slateblue3') +
  geom_bar(aes(y = offset_score), stat = 'identity', fill = 'red4') +
  geom_hline(yintercept = text_scores$offset[1], linetype = 'dashed', size = .5) +
  labs(title = 'Sentiment analysis: Game of Thrones, by character',
       y = 'Sentiment score') +
  coord_flip() +
  theme(axis.title.y = element_blank())
```


Sentiment analysis: Game of Thrones, by character
Eddard
Jon
Tyrion
Catelyn
Daenerys
Bran
Arya
Sansa
Prologue
−1000
−500
0
500
Sentiment score