Assignment 10A – Sentiment Analysis

Kevin Martin

2025-11-01

R Markdown

This analysis follows the workflow in *Text Mining with R*, Chapter 2 ("Sentiment analysis with tidy data") and then extends it to Week 10 discussion text.

Packages

```
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.2
## v ggplot2   3.5.2     v tibble    3.3.0
## v lubridate 1.9.4     v tidyr     1.3.1
## v purrr     1.1.0
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
library(tidytext)
library(janeaustenr)  # for the book example
```

Jane Austen Books

```
# Get Jane Austen books and make them tidy
austen <- janeaustenr::austen_books() %>%
  group_by(book) %>%
  mutate(linenumber = row_number()) %>%
  ungroup()

tidy_austen <- austen %>%
  unnest_tokens(word, text)

# Get a sentiment lexicon from tidytext
bing_lex <- get_sentiments("bing")
```

```
austen_sent <- tidy_austen %>%
  inner_join(bing_lex, by = "word") %>%
  count(book, sentiment) %>%
  tidyr::pivot_wider(names_from = sentiment,
                     values_from = n,
                     values_fill = 0) %>%
  mutate(net_sentiment = positive - negative)
```

```
## Warning in inner_join(., bing_lex, by = "word"): Detected an unexpected many-to-many relationship be
## i Row 435434 of `x` matches multiple rows in `y`.
## i Row 5051 of `y` matches multiple rows in `x`.
## i If a many-to-many relationship is expected, set `relationship =
##   "many-to-many"` to silence this warning.
```

```
austen_sent
```

```
## # A tibble: 6 x 4
##   book                negative positive net_sentiment
##   <fct>                  <int>    <int>         <int>
## 1 Sense & Sensibility     3671     4933          1262
## 2 Pride & Prejudice       3652     5052          1400
## 3 Mansfield Park          4828     6749          1921
## 4 Emma                    4809     7157          2348
## 5 Northanger Abbey        2518     3244           726
## 6 Persuasion              2201     3473          1272
```

corpus

```
# ---- corpus ----
my_posts <- tibble::tibble(
  source = c("Kevin_discussion", "Candace_post", "Joao_post", "Kevin_reaction"),
  text = c(
    "I described a use case where I had to process multiple JSON files from different folders.",
    "Candace talked about using Microsoft Graph JSON batching to reduce network roundtrips.",
    "Joao described e-commerce JSON files with product, customer, and revenue data.",
    "This workflow is really helpful but a little frustrating when the JSON is deeply nested."
  )
)

tidy_posts <- my_posts %>%
  unnest_tokens(word, text)
  # NOTE: we are NOT removing stop words yet because corpus is small

# ---- diagnostic: what matched bing? ----
tidy_posts %>%
  inner_join(get_sentiments("bing"), by = "word") %>%
  count(source, sentiment, word, sort = TRUE)
```

```
## # A tibble: 2 x 4
##   source         sentiment word            n
```

```
##   <chr>          <chr>     <chr>       <int>
## 1 Kevin_reaction negative  frustrating     1
## 2 Kevin_reaction positive  helpful         1
```

bing_posts

```
# all sources
all_sources <- tidy_posts %>%
  dplyr::distinct(source)

bing_posts <- tidy_posts %>%
  inner_join(get_sentiments("bing"), by = "word") %>%
  count(source, sentiment) %>%
  tidyr::pivot_wider(
    names_from = sentiment,
    values_from = n,
    values_fill = list(n = 0)
  ) %>%
  # make sure both columns exist
  { if (!"positive" %in% names(.)) dplyr::mutate(., positive = 0L) else . } %>%
  { if (!"negative" %in% names(.)) dplyr::mutate(., negative = 0L) else . } %>%
  dplyr::mutate(net_sentiment = positive - negative) %>%
  # join back to show sources with 0 matches
  right_join(all_sources, by = "source") %>%
  # replace NAs from right_join
  mutate(
    positive = tidyr::replace_na(positive, 0L),
    negative = tidyr::replace_na(negative, 0L),
    net_sentiment = tidyr::replace_na(net_sentiment, 0L)
  )

bing_posts
```

```
## # A tibble: 4 x 4
##   source           negative positive net_sentiment
##   <chr>               <int>    <int>         <int>
## 1 Kevin_reaction          1        1             0
## 2 Kevin_discussion        0        0             0
## 3 Candace_post            0        0             0
## 4 Joao_post               0        0             0
```

afinn_posts

```
# ---- afinn_posts ----
afinn_posts <- tidy_posts %>%
  inner_join(get_sentiments("afinn"), by = "word") %>%
  group_by(source) %>%
  summarise(afinn_score = sum(value), .groups = "drop")

afinn_posts
```

```
## # A tibble: 1 x 2
##   source         afinn_score
##   <chr>                <dbl>
## 1 Kevin_reaction           0
```

Loughran posts

```
# ---- loughran_posts ----
loughran_posts <- tidy_posts %>%
  inner_join(get_sentiments("loughran"), by = "word") %>%
  count(source, sentiment, sort = TRUE) %>%
  tidyr::pivot_wider(
    names_from = sentiment,
    values_from = n,
    values_fill = 0
  )

loughran_posts
```

```
## # A tibble: 1 x 2
##   source         negative
##   <chr>             <int>
## 1 Kevin_reaction        1
```

Notes

I first reproduced the sentiment example from *Text Mining with R*, Chapter 2 to make sure my setup was correct. I then created a small corpus from Week 10 discussion content (my post and two classmates' posts about JSON). I applied two different sentiment lexicons (`bing` and `afinn`) to show that the same text can get scored in slightly different ways depending on the dictionary. This matches the assignment instructions.

Summary & Reflection

This assignment helped me explore how different sentiment lexicons—Bing, AFINN, and Loughran—analyze text and how their results change depending on the type of language used. The Bing lexicon worked best for my short dataset since it contained a few subjective words like helpful and frustrating. The AFINN lexicon didn't return any matches because my text didn't include emotional words that carry numeric values, and the Loughran lexicon, which is designed for financial or business contexts, showed how domain-specific tools can be more suitable for certain topics.

Even though I reviewed the materials beforehand, I didn't think about how little emotional vocabulary my chosen text would include. Since my discussion post focused on processing JSON files, I thought it would be interesting to reuse that same topic for sentiment analysis. Once I ran it, I saw that most of the words were technical and neutral, which meant the results were limited and didn't produce visuals like the example bar chart in the textbook.

I briefly considered switching to a more emotional dataset just to get more colorful results, but the concept still worked—and I liked the connection to my discussion topic. This showed me that sentiment analysis depends heavily on the type of text being analyzed. Some topics, like data processing or technical writing, naturally contain less emotion, and that's fine—the key is understanding why that happens.

If I wanted to see a wider emotional range, I could have used:

Product or restaurant reviews (e.g., “The food was amazing, but the service was terrible.”)

Movie reviews or social media posts (e.g., “I love this show!” or “This update completely ruined it.”)

News articles or blog comments about topics that trigger opinions or emotions.

Overall, this project reminded me that sentiment tools aren’t one-size-fits-all; the results only make sense when the dataset fits the type of analysis.