Data 607 - Assignment 7 - Intro to Sentiment Analysis

William Jasmine

2022-11-10

Introduction

The code below is cited from Chapter 2 of *Welcome to Text Mining with R: A Tidy Approach* by Julia Silge and David Robertson, which shows how the `tidytext` library can be used to evaluate the sentiment of text. The example I have chosen shows how the `nrc` lexicon can be used in order to get the most commonly used words in Jane Austen novels that are associated with joy (text data used comes from the `janeaustenr` package).

```
# clean the Jane Austen text data
tidy_books <- austen_books() %>%
  group_by(book) %>%
  mutate(
    linenumber = row_number(),
    chapter = cumsum(str_detect(text,
                                regex("^chapter [\\divxlc]",
                                      ignore_case = TRUE)))) %>%
  ungroup() %>%
  unnest_tokens(word, text)

# get those words from the nrc lexicon mapped to "joy"
nrc_joy <- get_sentiments("nrc") %>%
  filter(sentiment == "joy")

# join the lexicon mappings to the Jane Austen book data.
tidy_books %>%
  filter(book == "Emma") %>%
  inner_join(nrc_joy) %>%
  count(word, sort = TRUE)
```

```
## Joining, by = "word"
```

```
## # A tibble: 301 x 2
##    word          n
##    <chr>     <int>
##  1 good        359
##  2 friend      166
##  3 hope        143
##  4 happy       125
##  5 love        117
##  6 deal         92
##  7 found        92
##  8 present      89
##  9 kind         82
```

```
## 10 happiness    76
## # ... with 291 more rows
```

As expected, the words shown above all exhibit a certain sense of joy.

The example above uses the `nrc` lexicon in order to perform this analysis, which maintains a dictionary of words that are mapped to different feelings or emotions. The `get_sentiments()` function can be used to show these mappings:

```
get_sentiments("nrc")
```

```
## # A tibble: 13,872 x 2
##    word        sentiment
##    <chr>       <chr>
##  1 abacus      trust
##  2 abandon     fear
##  3 abandon     negative
##  4 abandon     sadness
##  5 abandoned   anger
##  6 abandoned   fear
##  7 abandoned   negative
##  8 abandoned   sadness
##  9 abandonment anger
## 10 abandonment fear
## # ... with 13,862 more rows
```

In addition to `nrc` there are numerous other lexicons that can be used, each providing a slightly different methodology to perform sentiment analysis.

Sentiment Analysis Using Tweets

This section will use a different lexicon to analyze a number of Tweets that have been written concerning different entities. The data was pulled from Kaggle and was uploaded to Github. Each row of the data set pertains to a tweet about a different entity (i.e. Microsoft, Verizon, HomeDepot). The data has also been labelled to include the sentiment of each tweet (positive, negative, neutral) so that we can check the quality of our sentiment analysis. The data is downloaded and stored as a dataframe `tweets` in the code chunk below:

```
link <- getURL('https://raw.githubusercontent.com/williamzjasmine/CUNY_SPS_DS/master/DATA_607/Homeworks/
tweets <- read_csv(link, na=c("", "NA"))
```

```
## Rows: 74682 Columns: 4
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (3): entity, sentiment, tweet_content
## dbl (1): tweet_id
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
dim(tweets)
```

```
## [1] 74682     4
```

Based on the output above, we see that the `tweets` dataframe contains 74,682 tweets.

Data Cleaning

Before beginning the sentiment analysis, there are a number of data cleaning steps that need to be performed. To limit the scope of the data, the code chunk below filters `tweets` to include only those tweets concerning Facebook, Google, and Amazon. It also limits the only sentiments to "positive", "neutral", or "negative" (excluding "irrelevant").

```
tweets <- tweets %>%
  filter(entity %in% c('Amazon', 'Google', 'Facebook')) %>%
  filter(sentiment %in% c('Positive', 'Neutral', 'Negative'))
```

Because the data appears to include multiple tweets for the same `tweet_id`, the code below picks the first row for each `tweet_id`:

```
tweets <- tweets[!duplicated(tweets$tweet_id),]
dim(tweets)
```

```
## [1] 930   4
```

The output above shows that we have shrunk down the `tweets` data frame to now only include 930 tweets concerning three different entities. This will make the following sentiment analysis much more manageable.

Next, before tokenizing our data, the cell below pulls out the `tweet_id` and `sentiment` values for each row so that we can have them to compare to once the sentiment analysis is complete.

```
sentiments <- select(tweets, tweet_id, entity, sentiment)
sentiments <- sentiments %>%
  mutate(sentiment = tolower(sentiment))
tweets <- select(tweets, -sentiment, -entity)
```

Lastly, we need to modify the data set so that it contains a single row for each word present in the tweet (tokenization). This is done in the cell below using the `unnest_tokens` function:

```
tweets <- unnest_tokens(tweets, words, tweet_content)
colnames(tweets) <- c('tweet_id', 'word')
head(tweets)
```

```
## # A tibble: 6 x 2
##   tweet_id word
##      <dbl> <chr>
## 1        1 amazon
## 2        1 wtf
## 3        2 iâ
## 4        2 m
## 5        2 really
## 6        2 disappointed
```

The output above represents the final dataframe that we can now use to perform a sentiment analysis.

Sentiment Analysis

The lexicon that will be used for this example is the `afinn` lexicon, which contains 2,477 words rated on a scale from -5 to 5. In this case, the scale represents word's positivity/negativity, with higher scores indicating a more positive sentiment. Some of these mappings can be seen below, which are stored in the dataframe `afinn`:

```
afinn <- get_sentiments("afinn")
head(afinn)
```

```
## # A tibble: 6 x 2
##   word       value
##   <chr>      <dbl>
## 1 abandon       -2
## 2 abandoned     -2
## 3 abandons      -2
## 4 abducted      -2
## 5 abduction     -2
## 6 abductions    -2
```

The cell below joins the `afinn` data frame to our `tweets` data set in an attempt to get a sentiment score for each word.

```r
tweets <- tweets %>%
  left_join(afinn, by='word') %>%
    mutate(value_clean = ifelse(is.na(value), 0, value))
```

To determine the sentiment of the entire tweet, we can regroup our `tweets` dataframe and sum the sentiment scores of each word:

```r
tweets <- tweets %>%
  group_by(tweet_id) %>%
    summarise(
      score = sum(value_clean),
      num_words = n(),
      num_found_words = sum(ifelse(is.na(value) == FALSE, 1, 0))
    )
```

Lastly, given a sentiment score s and the number of scored words N, we can categorize the tweet as negative, neutral, or positive via the following conditions:

- Negative if: $\frac{s}{N} < -1$
- Neutral if: $-1 \leq \frac{s}{N} \leq 1$
- Negative if: $\frac{s}{N} > 1$

This is performed in R in the cell below:

```r
tweets <- tweets %>%
  mutate(sentiment_pred =
    case_when(
      score / num_found_words < -1 ~ "negative",
      score / num_found_words > 1  ~ "positive",
      TRUE                         ~ "neutral"
    )
  )
head(tweets)
```

```
## # A tibble: 6 x 5
##   tweet_id score num_words num_found_words sentiment_pred
##      <dbl> <dbl>     <int>           <dbl> <chr>
## 1        1    -4         2               1 negative
## 2        2    -4        44               2 negative
## 3        3    -2        30               2 neutral
## 4        4    -3        15               1 negative
## 5        6     1         3               2 neutral
## 6        7     0        11               0 neutral
```

Finally, we can check these newly made predictions to the ones that initially came with the data set:

```
tweets <- tweets %>%
  left_join(sentiments, by='tweet_id') %>%
    mutate(correct = ifelse(sentiment_pred == sentiment, TRUE, FALSE))

table(tweets$correct)
```

```
##
## FALSE  TRUE
##   519   409
```

Based on the output above, we can see that overall the sentiment analysis didn't do too great, but did outperform chance: it was correct about 44% of the time. This number gets better when you remove the neutral tweets and redo our original scoring by simply using a sum to determine if a tweet had a positive or negative sentiment:

```
tweets <- tweets %>%
  filter(sentiment != 'neutral') %>%
    mutate(
      correct_no_neutral = ifelse(
        (score > 0 & sentiment == "positive") |
        (score < 0 & sentiment == 'negative'), TRUE, FALSE)
    )

table(tweets$correct_no_neutral)
```

```
##
## FALSE  TRUE
##   158   298
```

When ignoring the neutral tweets, the sentiment analysis performed was correct a respectable 65% of the time.

Conclusion

Obviously this is not the most accurate sentiment analysis performed, and there are a number of improvements that could likely improve the final accuracy score. However, the preceding section provides a framework for how one might go about analyzing text in this way.