DATA607: Sentiment Analysis

Alexis Mekueko

10/31/2020

Github link: https://github.com/asmozo24/DATA607_Assignment10

Web link: https://rpubs.com/amekueko/684055

R Packages

```
#loading all library needed for this assignment
# this library are already in my Local downloaded_packages if not, I can install each
# install.packages("rtweet")
library(tidyverse)
library(DT)
library(knitr)

#library(plyr)
library(XML)
library(RCurl)
library(jsonlite)
library(httr)
library(tidytext)
```

```
## Warning: package 'tidytext' was built under R version 4.0.3
```

```
library(tidyr)
library(janeaustenr)
```

```
## Warning: package 'janeaustenr' was built under R version 4.0.3
```

```
library(textdata) # https://rdrr.io/cran/textdata/f/README.md
```

```
## Warning: package 'textdata' was built under R version 4.0.3
```

```
get_sentiments("afinn") #general purpose lexions from Finn Arup Nielsen, AFINN is a lexicon of English
```

```
## # A tibble: 2,477 x 2
##    word       value
##    <chr>      <dbl>
##  1 abandon       -2
```

```
##   2 abandoned     -2
##   3 abandons      -2
##   4 abducted      -2
##   5 abduction     -2
##   6 abductions    -2
##   7 abhor         -3
##   8 abhorred      -3
##   9 abhorrent     -3
##  10 abhors       -3
## # ... with 2,467 more rows
```

```r
library(wordcloud)
```

```
## Warning: package 'wordcloud' was built under R version 4.0.3
```

```r
library(tm)
```

```
## Warning: package 'tm' was built under R version 4.0.3
```

```
## Warning: package 'NLP' was built under R version 4.0.3
```

```r
library(reshape2)
library(syuzhet)
```

```
## Warning: package 'syuzhet' was built under R version 4.0.3
```

```r
library(rtweet)
```

```
## Warning: package 'rtweet' was built under R version 4.0.3
```

```r
library(corpus)
```

```
## Warning: package 'corpus' was built under R version 4.0.3
```

```r
#library(maps)
#library(dice)
# #library(VennDiagram)
# #library(help = "dice")
#ibrary(DBI)
#library(dbplyr)

# library(rstudioapi)
# library(RJDBC)
# library(odbc)
# library(RSQLite)
# #library(rvest)

library(readr)
#library(ggpubr)
```

```
#library(fitdistrplus)
#library(ggplot2)
#library(moments)
#library(qualityTools)
#library(normalp)
#library(utils)
#library(MASS)
#library(qqplotr)
#library(DATA606)
```

Description

This assignment of week 10 is about sentiment analysis. Sentiment analysis is the use of language processing, text analysis, computational linguistics, and biometrics to systematically identify, extract, quantify, and study affective states and subjective information. To explore this language, we will start by getting the primary example code from book, In Text Mining with R, chapter 2. We will extend this sample code to incorporate at leasr one additional sentiment lexicon. We will include the references at the last part.

Approach

We will start by replicating the sample code from the book.

Sentiment analysis with inner join

```
#Let's look at the words with a joy score from the NRC lexicon. What are the most common joy words in E

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) #Notice that we chose the name word for the output column from unnest_token
```

```
# Now that the text is in a tidy format with one word per row, we are ready to do the sentiment analysi
nrc_joy <- get_sentiments("nrc") %>%  #required download from Mohammad, Saif M. and Turney, Peter D.
  filter(sentiment == "joy") # let's use the NRC lexicon and filter() for the joy words

tidy_books %>%  #by grouping filter(), inner_join, the data frame with the text from the books for the
  filter(book == "Emma") %>%  #
  inner_join(nrc_joy) %>%
  count(word, sort = TRUE) # the most common joy words in Emma?
```

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

```
## # A tibble: 303 x 2
##    word        n
##    <chr>   <int>
##  1 good      359
##  2 young     192
##  3 friend    166
##  4 hope      143
##  5 happy     125
##  6 love      117
##  7 deal       92
##  8 found      92
##  9 present    89
## 10 kind       82
## # ... with 293 more rows
```

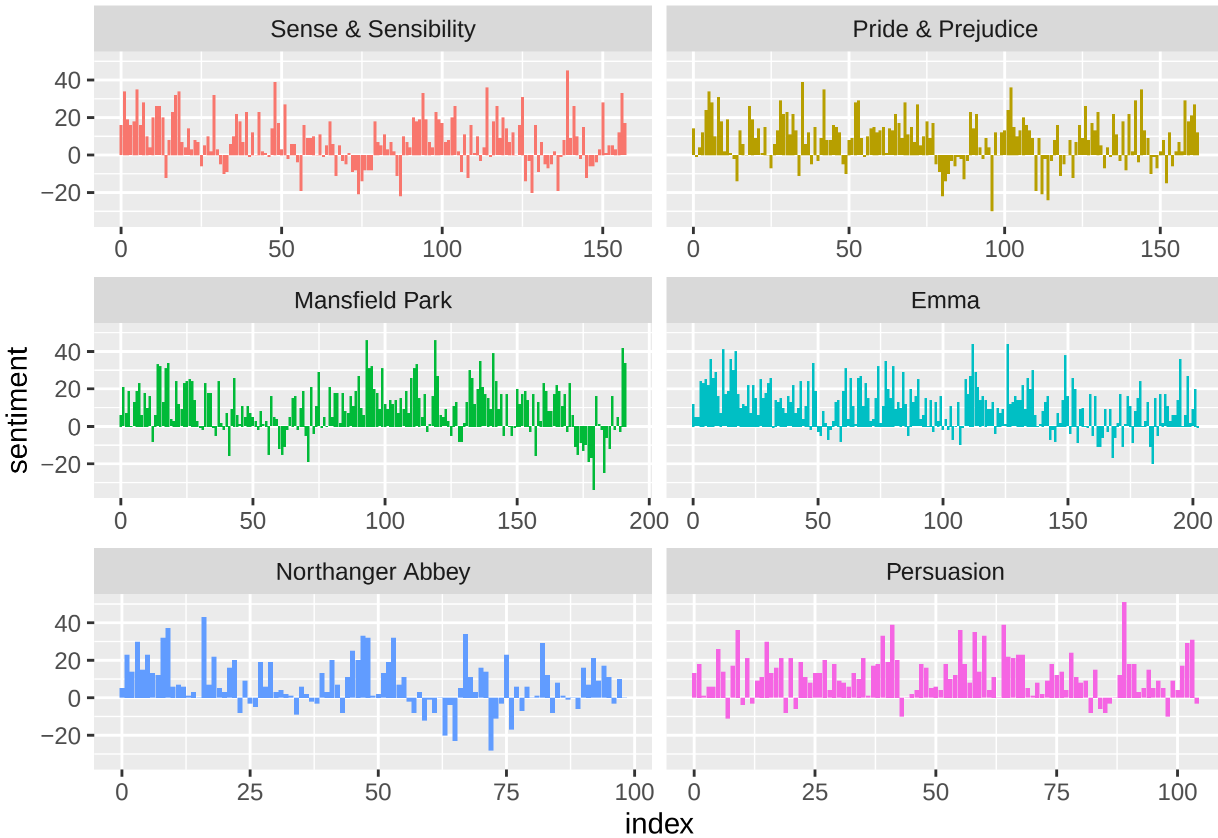
The %/% operator does integer division (x %/% y is equivalent to floor(x/y)) so the index keeps track of which 80-line section of text we are counting up negative and positive sentiment in. For these books, using 80 lines works well, but this can vary depending on individual texts, how long the lines were to start with, etc

```
# use spread() so that we have negative and positive sentiment in separate columns, and lastly calculat
jane_austen_sentiment <- tidy_books %>%
  inner_join(get_sentiments("bing")) %>%
  count(book, index = linenumber %/% 80, sentiment) %>%
  spread(sentiment, n, fill = 0) %>%
  mutate(sentiment = positive - negative)
```

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

Now we can plot these sentiment scores across the plot trajectory of each novel

```
# we are plotting against the index on the x-axis that keeps track of narrative time in sections of tex
ggplot(jane_austen_sentiment, aes(index, sentiment, fill = book)) +
  geom_col(show.legend = FALSE) +
  facet_wrap(~book, ncol = 2, scales = "free_x")
```




```
#We can see in Figure 2.2 how the plot of each novel changes toward more positive or negative sentiment
```

Comparing the three sentiment dictionaries

Let's use all three sentiment lexicons and examine how the sentiment changes across the narrative arc of Pride and Prejudice.

```r
#let's use filter() to choose only the words from the one novel we are interested in.
pride_prejudice <- tidy_books %>%
  filter(book == "Pride & Prejudice")
pride_prejudice
```

```
## # A tibble: 122,204 x 4
##    book              linenumber chapter word
##    <fct>                  <int>   <int> <chr>
##  1 Pride & Prejudice          1       0 pride
##  2 Pride & Prejudice          1       0 and
##  3 Pride & Prejudice          1       0 prejudice
##  4 Pride & Prejudice          3       0 by
##  5 Pride & Prejudice          3       0 jane
##  6 Pride & Prejudice          3       0 austen
##  7 Pride & Prejudice          7       1 chapter
##  8 Pride & Prejudice          7       1 1
```

```
##  9 Pride & Prejudice         10       1 it
## 10 Pride & Prejudice         10       1 is
## # ... with 122,194 more rows
```

we can use inner_join() to calculate the sentiment in different ways. Remember from above that the AFINN lexicon measures sentiment with a numeric score between -5 and 5, while the other two lexicons categorize words in a binary fashion, either positive or negative. To find a sentiment score in chunks of text throughout the novel, we will need to use a different pattern for the AFINN lexicon than for the other two.

```
afinn <- pride_prejudice %>%
  inner_join(get_sentiments("afinn")) %>%
  group_by(index = linenumber %/% 80) %>%  # use integer division (%/%) to define larger sections of te
  summarise(sentiment = sum(value)) %>%
  mutate(method = "AFINN")   #use the same pattern mutate() to find the net sentiment in each of these
```

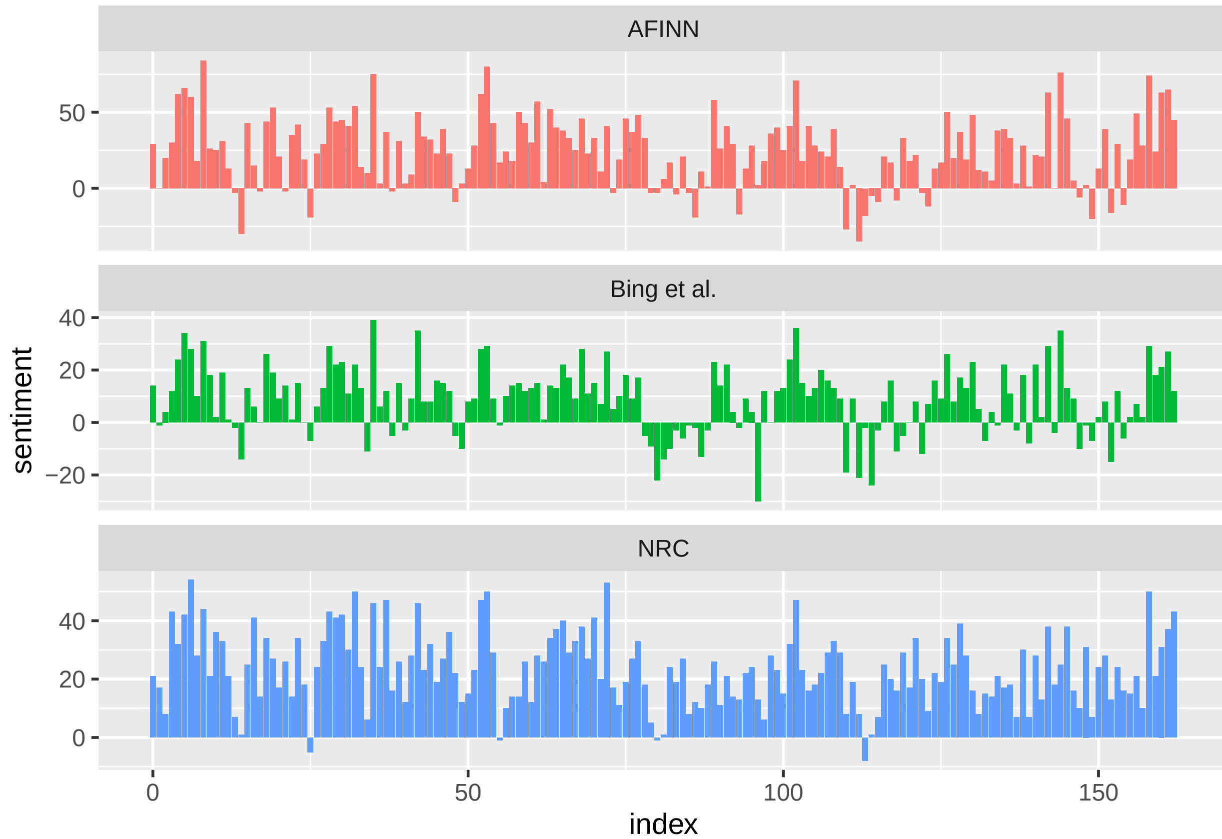
```
## Joining, by = "word"

## `summarise()` ungrouping output (override with `.groups` argument)
```

```
bing_and_nrc <- bind_rows(
  pride_prejudice %>%
    inner_join(get_sentiments("bing")) %>%
    mutate(method = "Bing et al."),  #use the same pattern mutate() to find the net sentiment in each o
  pride_prejudice %>%
    inner_join(get_sentiments("nrc") %>%
      filter(sentiment %in% c(
        "positive",
        "negative"
      ))) %>%
    mutate(method = "NRC")
) %>%
  count(method, index = linenumber %/% 80, sentiment) %>%
  spread(sentiment, n, fill = 0) %>%   #use the same pattern with count(), spread() to find the net sen
  mutate(sentiment = positive - negative)
```

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

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

We now have an estimate of the net sentiment (positive - negative) in each chunk of the novel text for each sentiment lexicon. Let's bind them together and visualize them in Figure 2.3.

```
bind_rows(
  afinn,
  bing_and_nrc
) %>%
  ggplot(aes(index, sentiment, fill = method)) +
  geom_col(show.legend = FALSE) +
  facet_wrap(~method, ncol = 1, scales = "free_y")
```



for example, the result for the NRC lexicon biased so high in sentiment compared to the Bing et al. result? Let's look briefly at how many positive and negative words are in these lexicons.

```
get_sentiments("nrc") %>%
  filter(sentiment %in% c(
    "positive",
    "negative"
  )) %>%
  count(sentiment)
```

```
## # A tibble: 2 x 2
##   sentiment     n
##   <chr>     <int>
## 1 negative   3324
## 2 positive   2312
```

```
get_sentiments("bing") %>%
  count(sentiment)
```

```
## # A tibble: 2 x 2
##   sentiment     n
##   <chr>     <int>
## 1 negative   4781
## 2 positive   2005
```

Most common positive and negative words

By implementing count() here with arguments of both word and sentiment, we find out how much each word contributed to each sentiment.

```
#By implementing count() here with arguments of both word and sentiment, we find out how much each word
bing_word_counts <- tidy_books %>%
  inner_join(get_sentiments("bing")) %>%
  count(word, sentiment, sort = TRUE) %>%
  ungroup()
```

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

This can be shown visually, and we can pipe straight into ggplot2, if we like, because of the way we are consistently using tools built for handling tidy data frames.

```
bing_word_counts %>%
  group_by(sentiment) %>%
  top_n(10) %>%
  ungroup() %>%
  mutate(word = reorder(word, n)) %>%
  ggplot(aes(word, n, fill = sentiment)) +
  geom_col(show.legend = FALSE) +
  facet_wrap(~sentiment, scales = "free_y") +
  labs(
    y = "Contribution to sentiment",
    x = NULL
  ) +
  coord_flip()
```

```
## Selecting by n
```

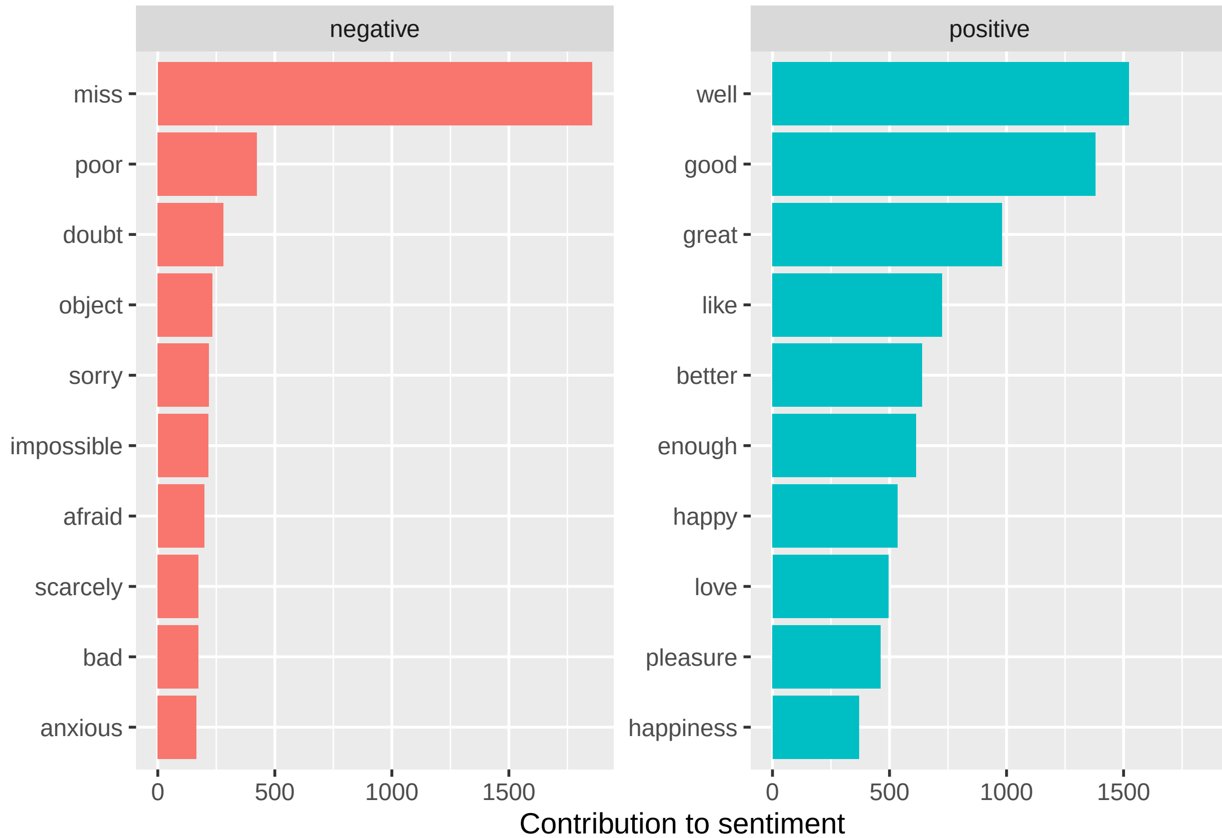


Figure 2.4: Words that contribute to positive and negative sentiment in Jane Austen's novels Figure 2.4 lets us spot an anomaly in the sentiment analysis; the word "miss" is coded as negative but it is used as a title for young, unmarried women in Jane Austen's works. If it were appropriate for our purposes, we could easily add "miss" to a custom stop-words list using bind_rows(). We could implement that with a strategy such as this.

```
custom_stop_words <- bind_rows(
  tibble(
    word = c("miss"),
    lexicon = c("custom")
  ),
  stop_words
)

custom_stop_words
```

```
## # A tibble: 1,150 x 2
##    word        lexicon
##    <chr>       <chr>
##  1 miss        custom
##  2 a           SMART
##  3 a's         SMART
##  4 able        SMART
##  5 about       SMART
##  6 above       SMART
##  7 according   SMART
```

```
##   8 accordingly SMART
##   9 across      SMART
##  10 actually    SMART
## # ... with 1,140 more rows
```

Worldcloud

We've seen that this tidy text mining approach works well with ggplot2, but having our data in a tidy format is useful for other plots as well.

For example, consider the wordcloud package, which uses base R graphics. Let's look at the most common words in Jane Austen's works as a whole again, but this time as a wordcloud in Figure 2.5.

```
tidy_books %>%
  anti_join(stop_words) %>%
  count(word) %>%
  with(wordcloud(word, n, max.words = 100))
```

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

```
## Warning in wordcloud(word, n, max.words = 100): house could not be fit on page.
## It will not be plotted.
```

Figure 2.5: The most common words in Jane Austen's novels

Let's do the sentiment analysis to tag positive and negative words using an inner join, then find the most common positive and negative words. Until the step where we need to send the data to comparison.cloud(), this can all be done with joins, piping, and dplyr because our data is in tidy format.

```
#In other functions, such as comparison.cloud(), you may need to turn the data frame into a matrix with

tidy_books %>%
  inner_join(get_sentiments("bing")) %>%
  count(word, sentiment, sort = TRUE) %>%
  acast(word ~ sentiment, value.var = "n", fill = 0) %>%
  comparison.cloud(
    colors = c("gray20", "gray80"),
    max.words = 100
  )
```

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

Figure 2.6: Most common positive and negative words in Jane Austen's novels The size of a word's text in Figure 2.6 is in proportion to its frequency within its sentiment. We can use this visualization to see the most important positive and negative words, but the sizes of the words are not comparable across sentiments.

Looking at units beyond just words

lots of useful work can be done by tokenizing at the word level, but sometimes it is useful or necessary to look at different units of text. For example, some sentiment analysis algorithms look beyond only unigrams (i.e. single words) to try to understand the sentiment of a sentence as a whole. These algorithms try to understand that "I am not having a good day." For these, we may want to tokenize text into sentences, and it makes sense to use a new name for the output column in such a case.

```
PandP_sentences <- tibble(text = prideprejudice) %>%
  unnest_tokens(sentence, text, token = "sentences")
```

#Let's look at just one.

```
PandP_sentences$sentence[2]
```

```
## [1] "however little known the feelings or views of such a man may be on his first entering a neighbou
```

The sentence tokenizing does seem to have a bit of trouble with UTF-8 encoded text, especially with sections of dialogue; it does much better with punctuation in ASCII. One possibility, if this is important, is to try using iconv(), with something like iconv(text, to = 'latin1') in a mutate statement before unnesting.

Another option in unnest_tokens() is to split into tokens using a regex pattern. We could use this, for example, to split the text of Jane Austen's novels into a data frame by chapter.

```
austen_chapters <- austen_books() %>%
  group_by(book) %>%
  unnest_tokens(chapter, text,
    token = "regex",
    pattern = "Chapter|CHAPTER [\\dIVXLC]"
  ) %>%
  ungroup()

austen_chapters %>%
  group_by(book) %>%
  summarise(chapters = n())
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## # A tibble: 6 x 2
##   book                chapters
##   <fct>                  <int>
## 1 Sense & Sensibility       51
## 2 Pride & Prejudice         62
## 3 Mansfield Park            49
## 4 Emma                      56
## 5 Northanger Abbey          32
## 6 Persuasion                25
```

We have recovered the correct number of chapters in each novel (plus an "extra" row for each novel title). In the austen_chapters data frame, each row corresponds to one chapter.

Near the beginning of this chapter, we used a similar regex to find where all the chapters were in Austen's novels for a tidy data frame organized by one-word-per-row. We can use tidy text analysis to ask questions such as what are the most negative chapters in each of Jane Austen's novels?

First, let's get the list of negative words from the Bing lexicon. Second, let's make a data frame of how many words are in each chapter so we can normalize for the length of chapters. Then, let's find the number of negative words in each chapter and divide by the total words in each chapter. For each book, which chapter has the highest proportion of negative words?

```r
bingnegative <- get_sentiments("bing") %>%
  filter(sentiment == "negative")

wordcounts <- tidy_books %>%
  group_by(book, chapter) %>%
  summarize(words = n())
```

```
## `summarise()` regrouping output by 'book' (override with `.groups` argument)
```

```r
tidy_books %>%
  semi_join(bingnegative) %>%
  group_by(book, chapter) %>%
  summarize(negativewords = n()) %>%
  left_join(wordcounts, by = c("book", "chapter")) %>%
  mutate(ratio = negativewords / words) %>%
  filter(chapter != 0) %>%
  top_n(1) %>%
  ungroup()
```

```
## Joining, by = "word"
## `summarise()` regrouping output by 'book' (override with `.groups` argument)
```

```
## Selecting by ratio
```

```
## # A tibble: 6 x 5
##   book                chapter negativewords words  ratio
##   <fct>                 <int>         <int> <int>  <dbl>
## 1 Sense & Sensibility      43           161  3405 0.0473
## 2 Pride & Prejudice        34           111  2104 0.0528
## 3 Mansfield Park           46           173  3685 0.0469
## 4 Emma                     15           151  3340 0.0452
## 5 Northanger Abbey         21           149  2982 0.0500
## 6 Persuasion                4            62  1807 0.0343
```

Extend the sample code from the book.

I am curious about what I discovered from the discussion board and I want to explore it. So, I will use my twitter account as corpus and lexions dataset_sentence_polarity sentence polarity dataset (Indicator for sentiment, "neg" for negative and "pos" for positive) , syuzhet (indicates sentiment scores and emotion) and some lexion above whenever possible.

Let's see if I could get to my twitter account (barely use it) and download my twiter archive While waiting for twitter to deliver the data, let's authenticate on twitter

```r
# al_twitter <- read.csv("")
```

Figure 7.1: All tweets from our accounts While waiting for twitter to deliver the data, let's authenticate on twitter API Well, even here, Thanks! We've received your request for API access and are in the process of reviewing it. So, let use some paper from today's papers on New York Times ..Time Running Short, Trump and Biden Return to Northern Battlegrounds… A Clash of Views Before Election Day

```r
# #login
# create_token(
# app = "your_app",
# consumer_key = "###",
# consumer_secret = "###",
# access_token = "###",
# access_secret = "###")
```

```r
# getting the txt file which has the NYT content about Time Running Short, Trump...
# text <- readLines(file.choose())
todayNews <- readLines("todayPaperNYT_election.txt", skip = 0)
todayNews1 <- VCorpus(VectorSource(todayNews))
```

Cleaning/Tidy todayNews1

```r
# let's remove "/", "@" and "|" and replace with space
toSpace <- content_transformer(function (x , pattern ) gsub(pattern, " ", x))
todayNews1 <- tm_map(todayNews1, toSpace, "/")
todayNews1 <- tm_map(todayNews1, toSpace, "@")
todayNews1 <- tm_map(todayNews1, toSpace, "\\|")
# Remove punctuations
todayNews1 <- tm_map(todayNews1, removePunctuation)
# Remove numbers
todayNews1 <- tm_map(todayNews1, removeNumbers)
# Remove english common stopwords
todayNews1 <- tm_map(todayNews1, removeWords, stopwords("english"))
# Eliminate extra white spaces
todayNews1 <- tm_map(todayNews1, stripWhitespace)
```

Analysis

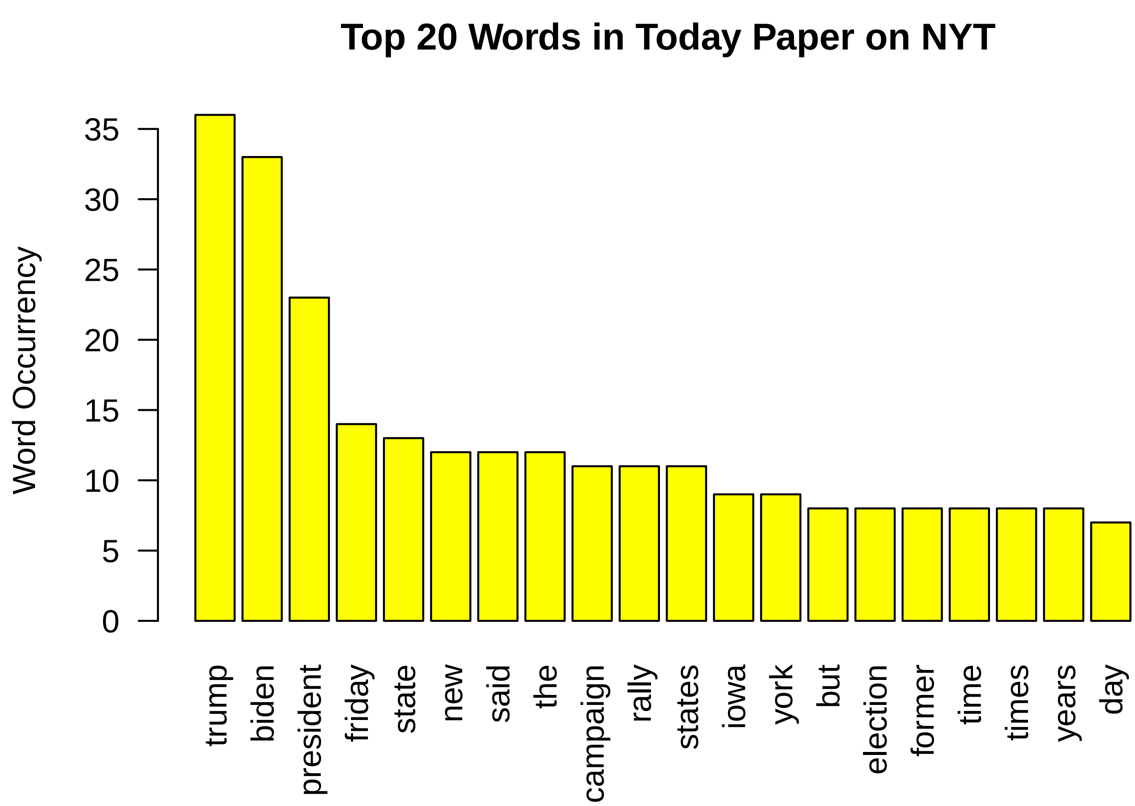
```r
# makking doc matrix

todayNews2 <- TermDocumentMatrix(todayNews1)
todayNews2M <- as.matrix(todayNews2)
# Sort by descearing value of frequency
todayNews2M <- sort(rowSums(todayNews2M),decreasing=TRUE)
todayNews2M <- data.frame(word = names(todayNews2M),freq=todayNews2M)
# let's view most the top 20 most frequent words
head(todayNews2M, 20)
```

```
##                   word freq
## trump            trump   36
## biden            biden   33
## president president   23
## friday          friday   14
## state            state   13
```

```
## new             new    12
## said           said    12
## the             the    12
## campaign   campaign    11
## rally         rally    11
## states       states    11
## iowa           iowa     9
## york           york     9
## but             but     8
## election   election     8
## former       former     8
## time           time     8
## times         times     8
## years         years     8
## day             day     7
```

Display some plot

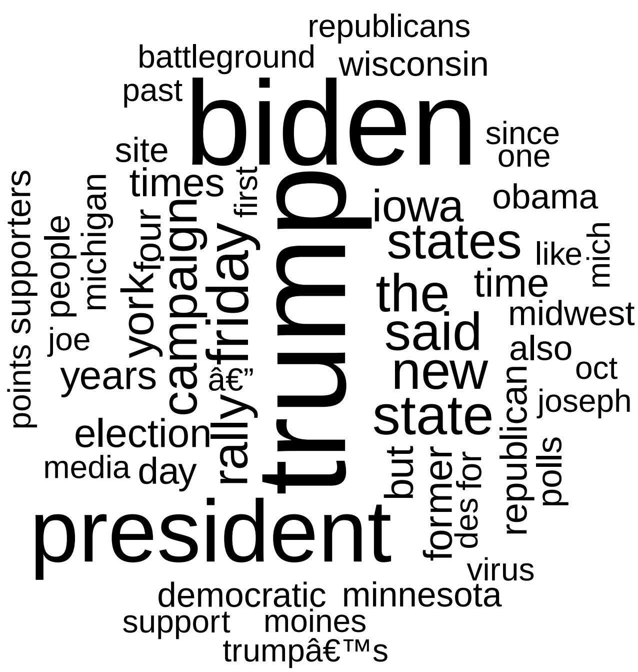
```r
#let's see plot for frequent words
barplot(todayNews2M[1:20,]$freq, las = 2, names.arg = todayNews2M[1:20,]$word,
        col ="yellow", main =" Top 20 Words in Today Paper on NYT",
        ylab = "Word Occurrency")
```



```r
# # let's remove some word that are not needed now...time, times, but, the , said
# todayNews1 <- tm_map(todayNews1, removeWords, c("time", "times", "but", "the", "said"))
# todayNews2 <- TermDocumentMatrix(todayNews1)
# todayNews2M <- as.matrix(todayNews2)
# # Sort by descearing value of frequency
# todayNews2M <- sort(rowSums(todayNews2M),decreasing=TRUE)
# todayNews2M <- data.frame(word = names(todayNews2M),freq=todayNews2M)
# #let's see plot for frequent words
# barplot(todayNews2M[1:20,]$freq, las = 2, names.arg = todayNews2M[1:20,]$word,
#         col ="yellow", main =" Top 20 Words in Today Paper on NYT",
#         ylab = "Word Occurrency")
```

Create cloud

```r
set.seed(343443)
wordcloud(words = todayNews2M$word, freq = todayNews2M$freq, min.freq = 5,
          max.words=100, random.order=FALSE, rot.per=0.40,
          )
```



Sentiment Score

```r
todayNews3 <- get_sentiment(todayNews, method="syuzhet")
# see the first row of the vector
head(todayNews3)
```

```
## [1] 0.0 0.6 0.0 0.0 0.0 0.0
```

```r
# see summary statistics of the vector
summary(todayNews3 )
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -5.00000  0.00000  0.00000  0.09139  0.00000  4.60000
```

My text file not good for syuzhet

Conclusion

We are very impressed by how deeply and powerful sentiment analysis can go. For instance, being able to go into a book and extract specific chapter and content is very intuitive. My text file not good for syuzhet because summary point to zero.

References

Silge, J. and Robinson, D. (2020). Text Mining with R: A Tidy Approach. Retrieved from https://www.tidytextmining.com.

Arnold, Taylor B. 2016. cleanNLP: A Tidy Data Model for Natural Language Processing. https://cran.r-project.org/package=cleanNLP.

Arnold, Taylor, and Lauren Tilton. 2016. coreNLP: Wrappers Around Stanford Corenlp Tools. https://cran.r-project.org/package=coreNLP.

Rinker, Tyler W. 2017. sentimentr: Calculate Text Polarity Sentiment. Buffalo, New York: University at Buffalo/SUNY. http://github.com/trinker/sentimentr.

This data was first used in Bo Pang and Lillian Lee, "Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales.", Proceedings of the ACL, 2005.

The Syuzhet lexicon, which includes afinnm bing and nrc lexicon, was developed in the Nebraska Literary Lab under the direction of Matthew L. Jockers. https://www.rdocumentation.org/packages/syuzhet