post02-Tiffany-Tsay

Tiffany Tsay

November 27, 2017

Textual Analysis with R

> How do we quickly grasp the important contents of a text? How can R break down and summarise large pieces of text?

Often times when given texts longer than a paragraph, most people must read the entire text or the introduction and conclusion in order to grap the contents of the text. When short on time, this is often inefficient, so there have been various ways to generate context and summaries. One of the most popular ones is to look and word frequencies in the text. But there are often difficulties in how to quickly generate these frequencies coming from a simple text file. We will explore some of the most efficient methods of finding word frequencies in texts using the packages `tidytext, dplyr, stringr`.

These are downloadable using the functions:
`install.packages("tidytext")`
`install.packages("dplyr")`
`install.packages("stringr")` `install.packages("ggplot2")`
We will also be installing a package that contains the 7 books for Harry Potter. `install_packages("bradleyboehmke/harrypotter")` Make sure to load these packages!

```r
library(dplyr)
library(tidytext)
library(stringr)
library(ggplot2)
library(harrypotter)
library(tidyr)
```

We will be experimenting with the syllabus that is provided with Stat133 at UC Berkeley. Make sure you download the text file by running:

```r
download.file("https://raw.githubusercontent.com/ucb-stat133/
              stat133-fall-2017/master/syllabus/policies.md",
              destfile = "FILEPATH.txt")
```

Remember to change FILEPATH to where you want the file to be downloaded to. Additionally, we will also be using

---

Transforming texts into data frames

The first function that is necessary to understand how to generate data frames for the sample texts would be: `unnest_tokens()`. As mentioned before, we'll be using the downloaded text file from earlier. We will use the scan function so that R can read the file and then save it into a vector. The argument file takes in the filepath, while the argument what takes in what kind of vector the file should be parsed into. In our case, because it is a text file, it should be character. We can then convert this vector into a data frame. Go ahead and check that the frequencies can be displayed properly using summary function.

```r
txt = scan(file = FILEPATH.txt, what = character())
txt_df = data.frame(txt)
head(txt_df)
```

```
## # A tibble: 6 x 2
##          txt stringsAsFactors
##        <chr>            <lgl>
## 1         ##            FALSE
## 2       Stat            FALSE
## 3        133            FALSE
## 4     Course            FALSE
## 5 Logistics             FALSE
## 6        and            FALSE
```

However, there are still some issues with what is being displayed in the data frame. We have words that are "different" when there is trailing punctuation (`hello` vs `hello.`). Similarly capitalization is preventing same words from being recognized (`hello` vs `Hello`). Now watch the magic that unnest_tokens presents!

```r
tidy_txt = txt_df %>%
  unnest_tokens(word, txt)
tidy_txt[,"word"]
```

```
## # A tibble: 2,649 x 1
##          word
##         <chr>
##   1      stat
##   2       133
##   3    course
##   4 logistics
##   5       and
##   6  policies
##   7       mrs
##   8    mutner
##   9     rules
## 10    images
## # ... with 2,639 more rows
```

As quick review, unnest_tokens will:

1. Remove punctuation
2. Split according to the argument `token = words/characters/sentences` (In our example it was default to word)
3. Converts to lowercase using argument `to_lower = TRUE/FALSE` (In our example it was default to TRUE)

Here is what the data frames would have looked like with splits according to sentences.

```
test_txt = txt_df %>%
  unnest_tokens(char, txt, token = "characters")
test_txt[,"char"]
```

```
## # A tibble: 12,663 x 1
##      char
##     <chr>
##   1     s
##   2     t
##   3     a
##   4     t
##   5     1
##   6     3
##   7     3
##   8     c
##   9     o
## 10     u
## # ... with 12,653 more rows
```

Frequency of Words

Now that our text is all set up and properly reformatted we can begin to reorder the text to see the frequency.
Use the `count` function to accomplish this.

```
tidy_txt %>%
  count(word, sort = TRUE)
```

```
## # A tibble: 846 x 2
##      word     n
##     <chr> <int>
##   1   the   128
##   2   you    87
##   3    to    78
##   4   and    58
##   5    of    54
##   6     a    45
##   7  your    42
##   8   for    41
##   9  will    40
## 10    be    37
## # ... with 836 more rows
```

Unfortunately, it becomes quite obvious that many of the top frequency words are words that do not give any insight into the content. Often times these are articles or pronouns. The tidytext package has called a large group of these “filler” words as `stop words`. We can then filter them out using the `anti-join` function.

```
info_txt = tidy_txt %>%
        anti_join(stop_words) %>%
        count(word, sort = TRUE)
```

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

```
info_txt
```

```
## # A tibble: 602 x 2
##           word     n
##          <chr> <int>
##  1       grade    20
##  2         lab    17
##  3    homework    15
##  4 assignments    12
##  5       class    12
##  6  assignment    11
##  7       final    11
##  8        time    11
##  9        labs     9
## 10      policy     9
## # ... with 592 more rows
```

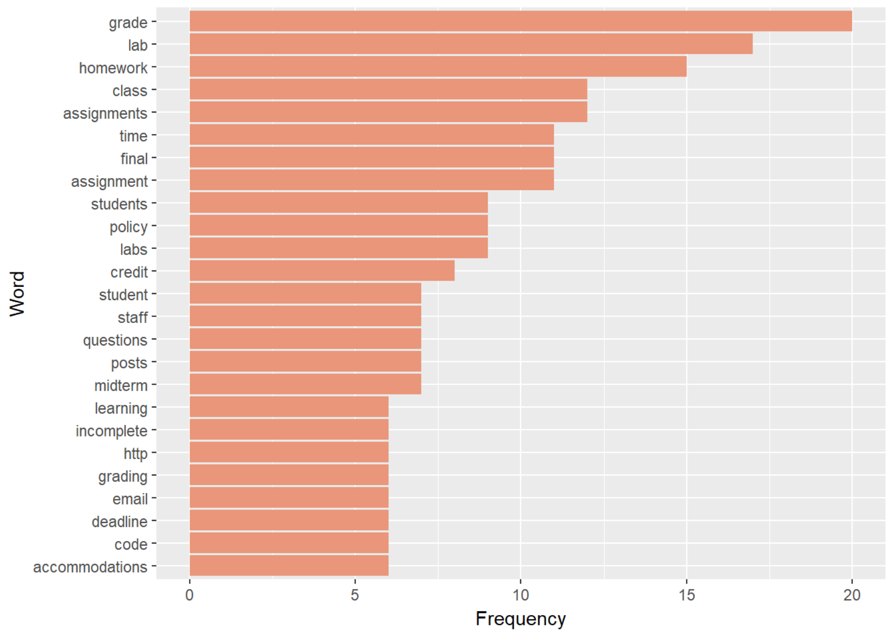
Now we have some more information on the actual content of the text. From here we can tell that clearly, our downloaded text has to do with education since there's many words like "homework", and "grade".

Transforming our data frames into visuals

Barchart

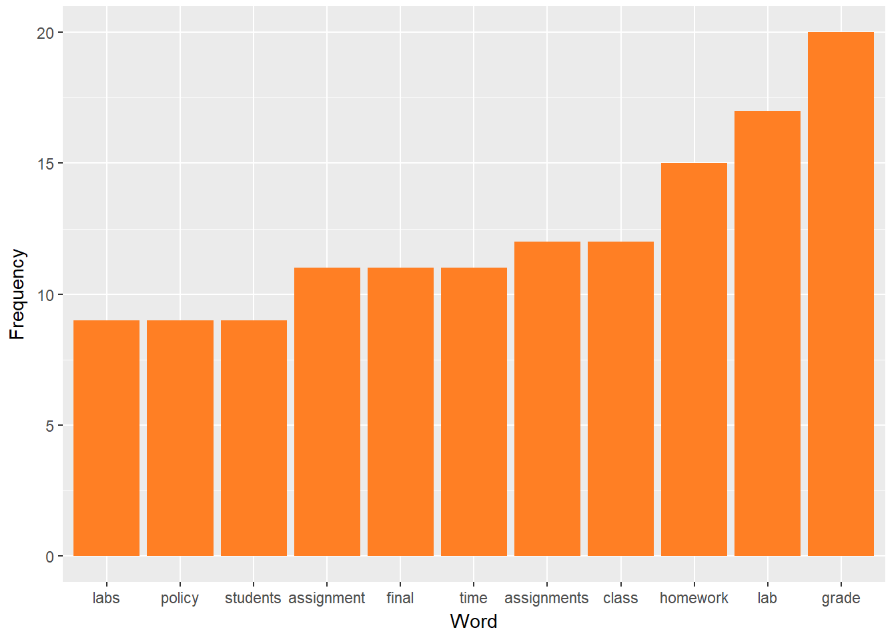
Now we can generate some visuals with this tidied up dataframe on the text.

```
info_txt %>%
  filter(n > 5) %>%
  mutate(word = reorder(word, n)) %>%
  ggplot(aes(word, n)) +
    geom_bar(stat = "identity", fill = "darksalmon") +
    labs(x = "Word", y = "Frequency") +
    coord_flip() +
    theme(legend.position="none")
```



We were able to generate this by use of ggplot2 library. You can adjust the filter to include or exclude fewer or more words. We also chose to flip the graph so that it would be visually easier to read across (left to right) for the words and how many times it was repeated. Here's what the graph would look like if we chose to show words that appeared at least nine times, and for the bars to be vertical.

```
info_txt %>%
  filter(n >8) %>%
  mutate(word = reorder(word, n)) %>%
  ggplot(aes(word, n)) +
    geom_bar(stat = "identity", fill = "chocolate1") +
    labs(x = "Word", y = "Frequency") +
    theme(legend.position="none")
```



As you can tell, this graph easily feels cluttered and uncomfortable if we were to have exhibited more words (lowered the filter), since the words would have less room to expand left to right.

Emotions of Texts

We will now be using the `harrypotter` package to investigate words across different books and the relationships across different books. One way we can explore how these books are similar and different are to look at the "emotion" words in the text. There are certain words that we naturally associate with certain emotions. For example, we associate happiness with: smile, joke, and friends. One way tidytext quantifies it is by assiging words a value between -5 to 5 based on their positive/negative sentiment. So let's take a look at what the general emotion of the Harry Potter series is.

There are different ways to look and score of the positivity/negativity. These three take slightly different approaches in quantifying the positive and negativie aspects.

1. get_sentiments("afinn")
2. get_sentiments("bing")
3. get_sentiments("nrc")

We first have to combine all 7 books in the Harry Potter package.

```
titles <- c("Philosopher's Stone", "Chamber of Secrets", "Prisoner of Azkaban",
            "Goblet of Fire", "Order of the Phoenix", "Half-Blood Prince",
            "Deathly Hallows")

books <- list(philosophers_stone, chamber_of_secrets, prisoner_of_azkaban,
           goblet_of_fire, order_of_the_phoenix, half_blood_prince,
           deathly_hallows)

series <- tibble()

for(i in seq_along(titles)) {

        clean <- tibble(chapter = seq_along(books[[i]]),
                        text = books[[i]]) %>%
             unnest_tokens(word, text) %>%
             mutate(book = titles[i]) %>%
             select(book, everything())

        series <- rbind(series, clean)
}
series
```

```
## # A tibble: 1,089,386 x 3
##                   book chapter    word
##                  <chr>   <int>   <chr>
##  1 Philosopher's Stone       1     the
##  2 Philosopher's Stone       1     boy
##  3 Philosopher's Stone       1     who
##  4 Philosopher's Stone       1   lived
##  5 Philosopher's Stone       1      mr
##  6 Philosopher's Stone       1     and
##  7 Philosopher's Stone       1     mrs
##  8 Philosopher's Stone       1 dursley
##  9 Philosopher's Stone       1      of
## 10 Philosopher's Stone       1  number
## # ... with 1,089,376 more rows
```

We have looped through all the books and created the data frame called `series` that has the tidied version of all words in the series. Now that we have finished our data preparation we can go ahead and look at some of the `sadness` words in the first book of Philosopher's Stone.

```
nrcjoy <- get_sentiments("nrc") %>%
  filter(sentiment == "sadness")

series %>%
  filter(book == "Philosopher's Stone") %>%
  inner_join(nrcjoy) %>%
  count(word, sort = TRUE)
```

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

```
## # A tibble: 221 x 2
##       word     n
##      <chr> <int>
##  1   harry  1213
##  2    dark    62
##  3   black    57
##  4    fell    46
##  5    lost    29
##  6     bad    28
##  7  mother    28
##  8 feeling    25
##  9     fat    21
## 10    kill    20
## # ... with 211 more rows
```

Similarly, we can also look at the `joy` words in the first book of Philosopher's Stone.

```
nrcjoy <- get_sentiments("nrc") %>%
  filter(sentiment == "joy")

series %>%
  filter(book == "Philosopher's Stone") %>%
  inner_join(nrcjoy) %>%
  count(word, sort = TRUE)
```

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

```
## # A tibble: 175 x 2
##       word     n
##      <chr> <int>
##  1    good    88
##  2   found    69
##  3   white    35
##  4   green    31
##  5  mother    28
##  6 feeling    25
##  7 finally    23
##  8    hope    22
##  9 smiling    20
## 10 pleased    19
## # ... with 165 more rows
```

We can quickly notice that there's a potential problem. The word "Harry" is being quantified as negative, but it's because of the main character's name! A permanent solution so that "Harry" is not counted as a negative word is to list it as a custom_stop_word. Due to the number of other things we need to explore, we will look at an alternative way of dealing with it. An alternative way to make sure this word does not pull too much weight in a quick summary of what was "sad" in the book, we can adjust how much of the book we examine by using the spread function. Using the function `count` in conjunction with `spread` we can adjust what we index by, in this case we chose to separate the net feelings (positive - negative) by each chapter.

```
harrypotterfeels <- series %>%
  inner_join(get_sentiments("bing")) %>%
  count(book, index = chapter, sentiment) %>%
  spread(sentiment, n, fill = 0) %>%
  mutate(sentiment = positive - negative)
```

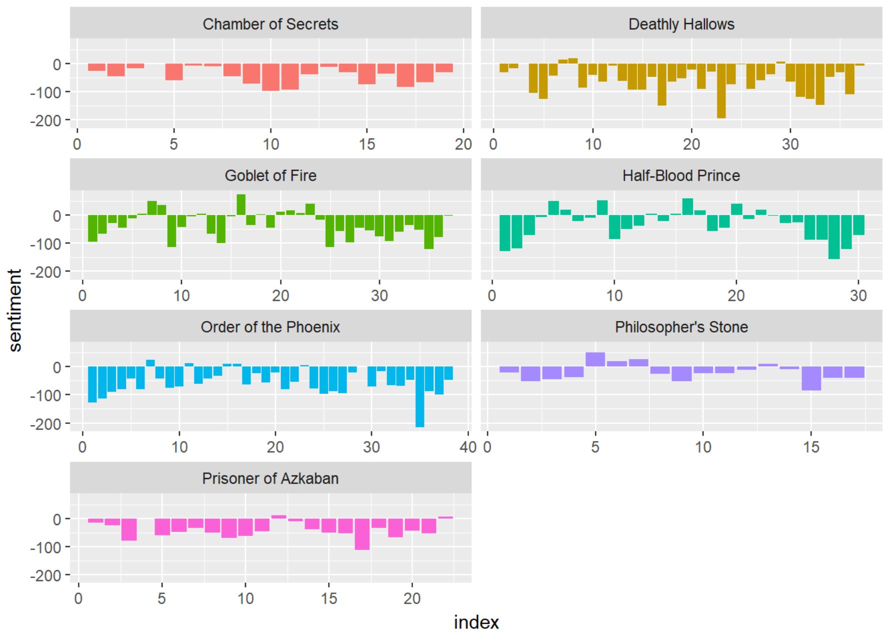
```
## Joining, by = "word"
```

```
harrypotterfeels
```

```
## # A tibble: 200 x 5
##                   book index negative positive sentiment
##                  <chr> <int>    <dbl>    <dbl>     <dbl>
##  1 Chamber of Secrets      1       99       74       -25
##  2 Chamber of Secrets      2      118       74       -44
##  3 Chamber of Secrets      3      131      114       -17
##  4 Chamber of Secrets      4      158      158         0
##  5 Chamber of Secrets      5      180      120       -60
##  6 Chamber of Secrets      6      151      145        -6
##  7 Chamber of Secrets      7      120      110       -10
##  8 Chamber of Secrets      8      184      140       -44
##  9 Chamber of Secrets      9      188      116       -72
## 10 Chamber of Secrets     10      221      124       -97
## # ... with 190 more rows
```

Now that we have calculated the net feelings for each chapter we can then plot it relative to every book for comparison.

```
ggplot(harrypotterfeels, aes(index, sentiment, fill = book)) +
  geom_col(show.legend = FALSE) +
  facet_wrap(~book, ncol = 2, scales = "free_x")
```



From this graph we can quickly tell that very few of the book chapters are net positive in feelings. And most of the story ark gets darker in the middle and end than the beginning. The quick visualization makes it very easy for readers to grasp the emotions of the book and the potential differences between the plot liens of each books.

What if we want to find out how much a word contributed to the overal feeling of the book? We can still create barcharts for the frequency and weight of each word. But first we have to tidy up and select only the correct feeling we want to look at.

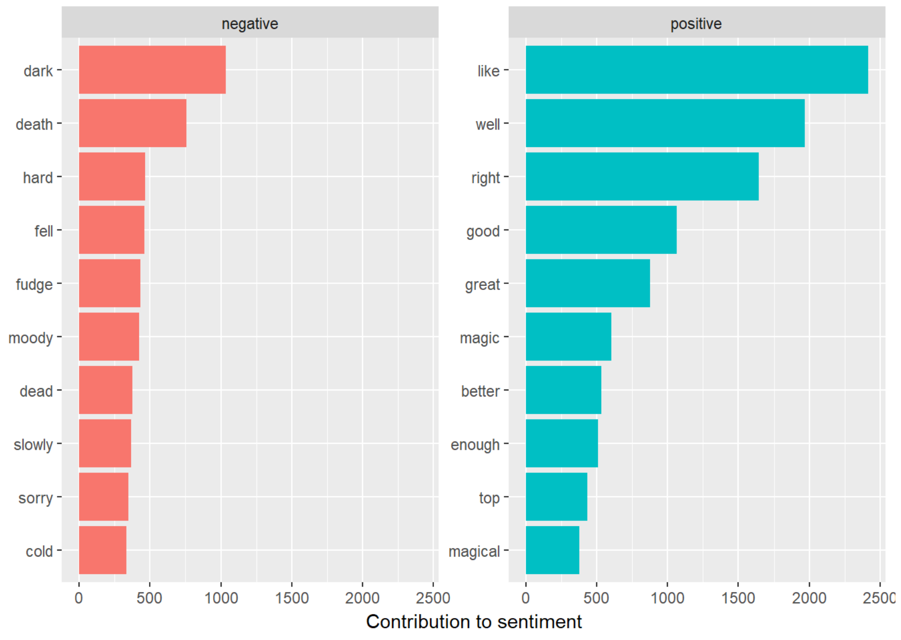
```
feels_word_counts <- series %>%
  inner_join(get_sentiments("bing")) %>%
  count(word, sentiment, sort = TRUE) %>%
  ungroup()
```

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

Now we're ready to graph!

```
feels_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
```



Upon looking we can see that there are many words that contribute to “positive” feelings, but they are not that strong relative to the negative words. It’s also interesting to notice that the positive words are not always exclusively postive, “like” is often used in similes.

What’s important is to realize that R is very powerful and can extract data from **anything**, not just your typical csvs. It’s very important to be able to analyze text files quickly, and not just csvs. Often times text files are much more itme consuming to read through, while CSV’s can quickly generate their own summaries (due to their quantitives nature). While there is no guaranteed way to perfectly summarise a text file, we have many ways of examining from trying to capture key words by freqency and overall emotions of the text.

Resources

Tidy Word Vectors part2

Tidy Word Vectors part1

Text Mining with R

Text Mining and Creating Tidy Text

Text Mining

Introduction to Text Analysis

Basic Text Mining