post01-jessica-yu

Jessica Yu

10/28/2017

Introduction to Text and Sentiment Analysis

Analysts, vendors, and users alike have estimated that roughly **80% of the world's data exists in unstructured form**. To provide a precise definition, data in unstructured form is data that does not have a well-defined data model, is typically text-heavy, and may include dates, numbers, and facts. These elements produce irregularities that render the data difficult to interpret using traditional programs.

So far in STAT 133, traditional R programs have been highly capable and useful in analyzing the 2017 NBA player data set as well as a few additional data sets. Over the past months, I've learned some basics of data cleaning, the beginnings of data analysis, and the many frustrations that come with learning R. In between poring over error messages and Piazza posts, I've noticed that much of the lab work and homework assignments has focused on cleaning numerical data. This is intuitive, since the NBA player dataset has many quantitative variables in comparison to categorical variables. Rather than working with quantitative and categorical variables, I'd like to explore **text data**.

The **objectives** of the post include:

> - Learn what text analysis is used for
> - Identify packages and functions that are useful in text analysis
> - Walk through the basic steps of text analytics
> - Understand how sentiment analysis is used in relation to text analysis
> - See the current widespread application of text and sentiment analysis/analytics

---

Background

At its core, text analytics is the exploration of text using a computer. There are two large motivators driving the importance of text analysis: digital texts and social media.

In recent years, the increased usage of laptops and the introduction of tablets has caused digital reading to proliferate and along with it the demand for digital texts. There are now countless e-books, with Project Gutenberg providing 54,000 titles online. Just as books have expanded into the digital realm, so have newspapers. Today, big and small news companies each have their own websites and Facebook pages, leading us to the second driver of text analysis, social media.

Social media– probably the biggest reason why we are on our phones these days because it is certainly not to make calls. Social media feeds include a large quantity of unstructured text data, including Facebook comments and tweets. By analyzing the contents of social media data, it is possible, for example, to draw meaningful conclusions on user sentiment across the readership of different news websites.

Some other **usages of text analytics** include:

- Risk management: present in all industries to mitigate the chance of business failure and practiced heavily in the financial industry
- Knowledge management: focuses on organizing extremely large amounts of information held by a company
- Cybercrime prevention: reduces Internet crime, which is widespread due to the anonymous, open communication nature of the Internet
- Contextual advertising: allows for smart advertising that tailors advertising campaigns to the interests and needs of the target market
- Business intelligence: helps analysts at large companies design business solutions and make smart business decisions
- Spam filtering: increases the effectiveness of spam filtering methods, including methods used to recognize spam emails
- Social media data analysis: extracts opinions and emotions associated with brands and products

---

Useful Packages

The tidytext package

The tidytext package provides highly useful tools and functions for working with text data. The package facilitates word processing and sentiment analysis for **tidy data**. Tidy data has a few characteristics:

- Each variable is a column
- Each observation is a row
- Each type of observational unit (words, sentences) is a table

The tidy text format is described as a table with one-token-per-row. A **token** is a unit of text, such as a word, that we are interested in analyzing. **Tokenization** is the process of splitting text into tokens. For instance, we may be splitting text into single words or into sentences.

Tokenization transforms the way text data is stored. Without using the tidytext package, R users often store text as strings, which may contain several words or multiple sentences within each string. This makes it difficult to analyze individual words or sentences. Through tidy text mining, just one token is stored in each row of a table.

In the following section, we will go through a simple example of text mining, text analytics, and sentiment analysis. To emphasize the importance of the tidytext package in facilitating working with text data, we will first clean the data without using tidytext. Afterwards, we will see how tidytext produces the same results much more efficiently.

The wordcloud package

The wordcloud package is useful for producing quick visualizations of text data. The package generates wordclouds that display words with sizes relative to their frequency. While wordclouds are not useful for many forms of data analysis involving quantitative variables, they can easily show you the words in text data that crop up the most.

A Walk Through of Basic Text Analytics

The post will walk through a simple text and sentiment analysis/analytics example using data from 155 comments posted in response to a New York Times article, Evidence of a Toxic Environment for Women in Economics.

Background context

Alice Wu is a recent economics graduate of UC Berkeley who wrote her senior thesis on gender stereotyping in the economics field by analyzing posts made on a forum, Economics Job Market Rumors. She uncovered that many of the anonymous posts on the forum were misogynistic and otherwise biased against women, describing them based on physical and non-professional characteristics compared with their male counterparts.

By analyzing the comments on the New York Times article, we can perhaps begin to gain a sense of how New York Times readers feel about Alice Wu's thesis and more broadly, how readers reacted to gender disparities in the economics field.

1. Get the Data:

R can help its users extract data from the web through many different ways. In this post, we'll focus on reading plain (raw) text using readLines().

As discussed in lecture, readLines() is one of R's basic reading functions that reads each line of plain text as a character string. This results in an output element with a number of elements equal to the number of lines in the file.

```r
# load packages
library(ggplot2)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
library(tidytext)
library(wordcloud)
```

```
## Loading required package: RColorBrewer
```

```r
nyt <- readLines("../data/nyt-comments.txt")
head(nyt)
```

```
## [1] "This is a little unrelated (but I was an academic economist for 2 decades and I want to comment on somethi
ng)."
## [2] "Every profession has \"insider jokes.\" Here's my favorite for the Econs:"

## [3] "\"An Economist is a man who knows 100 ways to make love to a woman - but doesn't know any women.\""

## [4] "(If you don't get it you never took Econ.)"

## [5] "Hello moderators:"

## [6] ""
```

```r
length(nyt)
```

```
## [1] 423
```

Note that the comments include capitalization and punctuation that make the text difficult to work with. We will address this issue in Step 2 of this example. For now, also observe that there are 423 lines of output corresponding to the 423 lines contained in the .txt file.

2. Clean the Data:

Now that the text file has been loaded into R, we can remove nonessential characters such as punctuation, numbers, web addresses, etc., so that we can eventually process just the words used in the comments.

tolower()

The function **tolower()** turns characters contained in character vectors into lower case. It has just one argument, the character vector.

gsub()

The function **gsub** is used for **pattern matching and replacement**. Essentially, gsub recognizes and replaces old strings with new strings that we want instead.

We will use gsub(pattern = "[^[:alnum:][:space:]']", replacement = "", nyt). To break down what this means piece by piece, let's begin with some more details on the arguments of the gsub() function.

From the **help documentation**:

> gsub(pattern, replacement, x, ignore.case = FALSE, perl = FALSE, fixed = FALSE, useBytes = FALSE)

- pattern: the string that is to be replaced
- replacement: the new string to replace the original string specified in the pattern argument
- x: string we're working with that contains the pattern string

For text analysis, we would like to eliminate all non-alphanumeric and non-space characters. In other words, we want to **remove all punctuation, symbols, etc. while keeping letters, numbers, and spaces**.

We begin by setting pattern = "[^[:alnum:][:space:]']" :

- '[:alnum:]'
  A list of all alphanumeric characters; a convenient combination of the lists '[:alpha:]' and '[:digit:]'

- '[:space:]'
  A list of space characters

We need to add '^' in front of the lists [:alnum:][:space:]. '^' will match any character *not* included in the lists enclosed in brackets. Thus, gsub will match all non-alphanumeric and non-space characters, which is precisely what we want.

In case you're wondering about the abundance of brackets and colons:

- A bracket expression is, as its name suggests, a list of characters enclosed by '[' and ']'. **It matches any character included in the list**.

- '[:' The open character class symbol, and should be followed by a valid character class name such as alnum, space, lower, punct.

- ':]' The close character class symbol.

Next, we set replacement = "", indicating that the characters specified in the pattern should be replaced with nothing (aka removed).

Finally, we specify x = nyt so that gsub knows we're working with the character strings contained in nyt.

```
nyt <- tolower(nyt) # convert all characters to lowercase
clean_nyt <- gsub("[^[:alnum:][:space:]']", "", nyt) # remove anything that's not an alphanumeric character, a spa
ce, or apostrophe (as seen right after the space character list) from nyt character strings
head(clean_nyt)
```

```
## [1] "this is a little unrelated but i was an academic economist for 2 decades and i want to comment on somethin
g"
## [2] "every profession has insider jokes here's my favorite for the econs"

## [3] "an economist is a man who knows 100 ways to make love to a woman  but doesn't know any women"

## [4] "if you don't get it you never took econ"

## [5] "hello moderators"

## [6] ""
```

```
length(clean_nyt) # there are 423 strings
```

```
## [1] 423
```

One last thing: Because we want to analyze every word contained in the comments, we need to separate our current strings that contain multiple words into strings that each contain only one word. We will use the functions paste() and strsplit().

paste()

The paste function is used to **concatenate strings** after converting them to characters and separating them by a designated separator.

From the **help documentation**:

> paste (…, sep = " ", collapse = NULL)

- … R objects to be converted to character vectors
- sep = a character string to separate the terms, " " by default

By calling paste(tidy_nyt, sep = " ", collapse =" "), we first get rid of the 423 individual lines by separating the contents of tidy_nyt by blank spaces, and then we collapse them into one character vector with the words separated by spaces.

```
paste(clean_nyt, sep= " ", collapse = " ")
```

strsplit()

The function strsplit() is used to **split the elements of a character vector into shorter substrings**, based on the character object that we specify as the separator (think commas in csv files).

From the **help documentation**:

> strsplit(x, split, fixed = FALSE, perl = FALSE, useBytes = FALSE)

- x: the character vector, each element of which is to be split

- split: the character vector or object containing expressions to use for splitting.

We can now split the giant character string above into 9595 individual strings representing the 9595 words used in the comments.

```
nyt_split <- strsplit(paste(clean_nyt, sep= " ", collapse = " "), ' ')[[1]]
head(nyt_split)
```

```
## [1] "this"      "is"        "a"         "little"    "unrelated" "but"
```

```
length(nyt_split) # There are 9595 individual words
```

```
## [1] 9595
```

Now we can analyze each individual word used in the comments.

2.1 Using tidytext to clean the data

unnest_tokens()

The unnest_tokens() function is contained in the tidytext package. It **splits a column from a data frame into tokens**. The splitting is done using the tokenizers package, ultimately allowing unnest_tokens() to **produce a table with one-token-per-row**.

From the **help documentation**:

> unnest_tokens(tbl, output, input, token = “words”, format = c(“text”, “man”, “latex”, “html”, “xml”), to_lower = TRUE, drop = TRUE, collapse = NULL, …)

- tbl: a data frame
- output: output column to be created
- input: input column that gets split

```
df_nyt <- data_frame(comments = nyt)
df_nyt
```

```
## # A tibble: 423 x 1
##                                                                          comments
##                                                                             <chr>
##  1 this is a little unrelated (but i was an academic economist for 2 decades a
##  2 "every profession has \"insider jokes.\" here's my favorite for the econs:"
##  3 "\"an economist is a man who knows 100 ways to make love to a woman - but d
##  4                                  (if you don't get it you never took econ.)
##  5                                                         hello moderators:
##  6
##  7
##  8 "why are you still sitting on my comment of getting on for 24 hours ago? tr
##  9
## 10
## # ... with 413 more rows
```

```
tidy_nyt <- unnest_tokens(tbl = df_nyt, output = words, input = comments)
tidy_nyt
```

```
## # A tibble: 9,424 x 1
##        words
##        <chr>
##  1      this
##  2        is
##  3         a
##  4    little
##  5 unrelated
##  6       but
##  7         i
##  8       was
##  9        an
## 10  academic
## # ... with 9,414 more rows
```

Note that tidytext enabled us to remove capitalization, eliminate punctuation, and separate strings containing multiple words all in one efficient step.

3. Word count

We can use a simple word count to identify the words that cropped up the most frequently in the comments. First, we need to convert tidy_nyt from a character vector into a data frame.

```
nyt_df <- data.frame(word = nyt_split, stringsAsFactors = FALSE) # create a data frame with a column named word, t
hat contains the words used in the comments
head(nyt_df)
```

```
##         word
## 1       this
## 2         is
## 3          a
## 4     little
## 5 unrelated
## 6        but
```

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

```
## # A tibble: 2,220 x 2
##      word     n
##     <chr> <int>
##  1   the   413
##  2    of   264
##  3    to   259
##  4         241
##  5     a   218
##  6   and   204
##  7  that   184
##  8    is   180
##  9    in   157
## 10     i   148
## # ... with 2,210 more rows
```

Naturally, articles in grammar and other “filler” words have the highest counts. Fortunately, R has a **data frame of common English stop words** (such as “a”, “about”, “you”), with 1149 words that definitely includes most of the filler words we want to remove. To **eliminate the filler words** that are not useful to text analysis, we use **anti_join** from the dplyr package with the stop_words data frame.

```
head(stop_words)
```

```
## # 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
```

```
nrow(stop_words)
```

```
## [1] 1149
```

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

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

```
## # A tibble: 1,781 x 2
##          word     n
##         <chr> <int>
##  1              241
##  2      women    94
##  3  economics    42
##  4     people    30
##  5 economists    26
##  6     female    24
##  7      study    24
##  8      field    22
##  9       male    19
## 10        job    18
## # ... with 1,771 more rows
```

```
nrow(nyt_df) # There are now just 1781 key words compared to 9595 total words
```

```
## [1] 9595
```

Now we see that some of the most frequently used words include “women”, “female”, “male”, indicating that gender heavily influences the discussion. Meanwhile, the words “study”, “field”, and “job” also crop up frequently, suggesting *there may be some conversation of the relation between gender and professional careers in economics*.

4. **Sentiment Analysis**

To make some more meaningful observations as related to the **opinions and emotions** of the commenters, we can perform sentiment analysis upon the words used in comments. Conveniently, R has a lexicon called NRC featuring the NRC **sentiment categories** of anger, anticipation, disgust, fear, joy, sadness, surprise, or trust. This data frame has 27,314 rows containing words that are sorted into these sentiment categories.

```
head(sentiments)
```

```
## # A tibble: 6 x 4
##         word sentiment lexicon score
##        <chr>     <chr>   <chr> <int>
## 1     abacus     trust     nrc    NA
## 2    abandon      fear     nrc    NA
## 3    abandon  negative     nrc    NA
## 4    abandon   sadness     nrc    NA
## 5  abandoned     anger     nrc    NA
## 6  abandoned      fear     nrc    NA
```

Note that **different lexicons, such as "bing" and "loughran" have different sentiment categories**. Bing simply sorts words into positive and negative, while loughran contains sentiments such as litigious, uncertainty, constraining, and superfluous.

Let's sort the words used in the comments into sentiment categories to see which sentiments were the most dominant based on the nrc lexicon. We'll use the **get_sentiments()** function from the **tidytext package** to obtain nrc word and sentiment data in the form of a data frame. We will also use the dplyr package and its functions left_join() and filter().

```
nrc_breakdown <-
  nyt_df  %>%
        left_join(get_sentiments("nrc")) %>% # returns a data frame with two columns, word and sentiment. word con
tains all words in nyt_df, sentiment shows NA where words used in comments are not recognized/included in the nrc
lexicon. Sentiment words that do not exist in the comments are removed from the joined table.
        filter(!is.na(sentiment)) %>% # Selects only the rows that have word matches between nyt_df and the nrc le
xicon, removing the rows where the sentiment column has NA values.
        count(sentiment, sort = TRUE) # counts the number of words that match each sentiment, sorted in descending
order
```

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

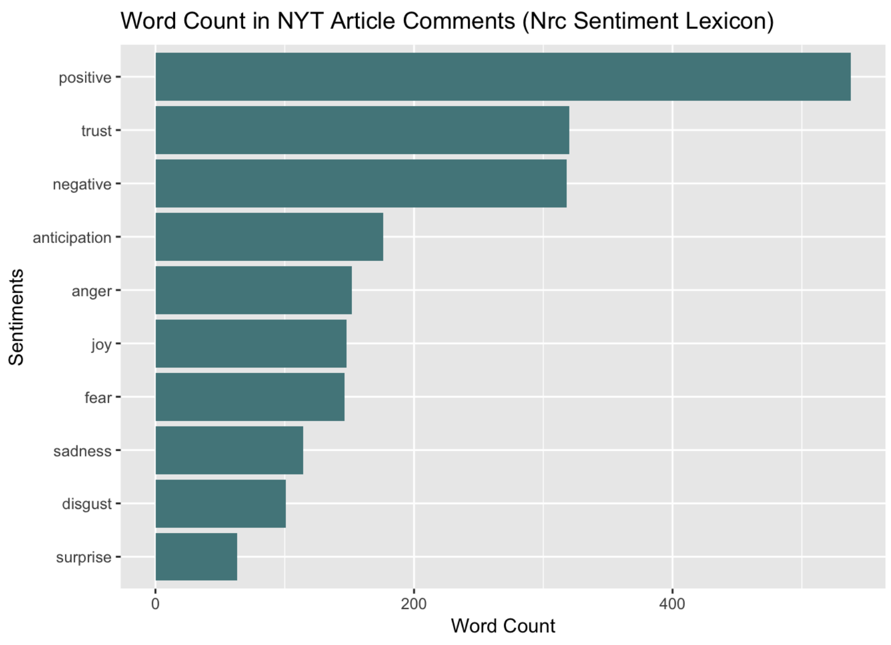
```
nrc_breakdown
```

```
## # A tibble: 10 x 2
##       sentiment     n
##           <chr> <int>
##  1     positive   538
##  2        trust   320
##  3     negative   318
##  4 anticipation   176
##  5        anger   152
##  6          joy   148
##  7         fear   146
##  8      sadness   114
##  9      disgust   101
## 10     surprise    63
```

Note: some of the words used in the comments are not contained in the nrc lexicon, and therefore cannot be matched to an nrc sentiment category. After appling the **left_join() function**, we remove these unmatched comment words using filter(), as they do not help us obtain any sentiment data from the nrc lexicon. We are left with just the comment words that do match nrc lexicon words and their corresponding sentiments. For instance, "love" is associated with 2 sentiments, joy and positive. "Love" will count towards both of these sentiment categories.

```
gg_wc_nrc <- ggplot(data = nrc_breakdown, aes(x = reorder(sentiment, n), y = n)) +
  geom_bar(stat = 'identity', fill = "cadetblue4") +
  labs(x = "Sentiments" , y = "Word Count", title = "Word Count in NYT Article Comments (Nrc Sentiment Lexicon)")
+
  coord_flip()

gg_wc_nrc
```



```
# ggsave("../images/word_count_nrc.pdf", gg_wc_nrc)
```

It appears that many of the words used by commenters were associated with positive, trustful, and negative sentiments. The strong presence of positive and trustful words may suggest agreement and praise for Alice Wu's ideas by New York Times readers. At the same time, the abundance of negative words implies that there may be debate and disagreement between commenters.

Let's check these findings against the Bing lexicon, which divides words into negative and positive sentiments.

```
bing_breakdown <-
  nyt_df  %>%
        left_join(get_sentiments("bing")) %>%
        filter(!is.na(sentiment)) %>%
        count(sentiment, sort = TRUE)
```

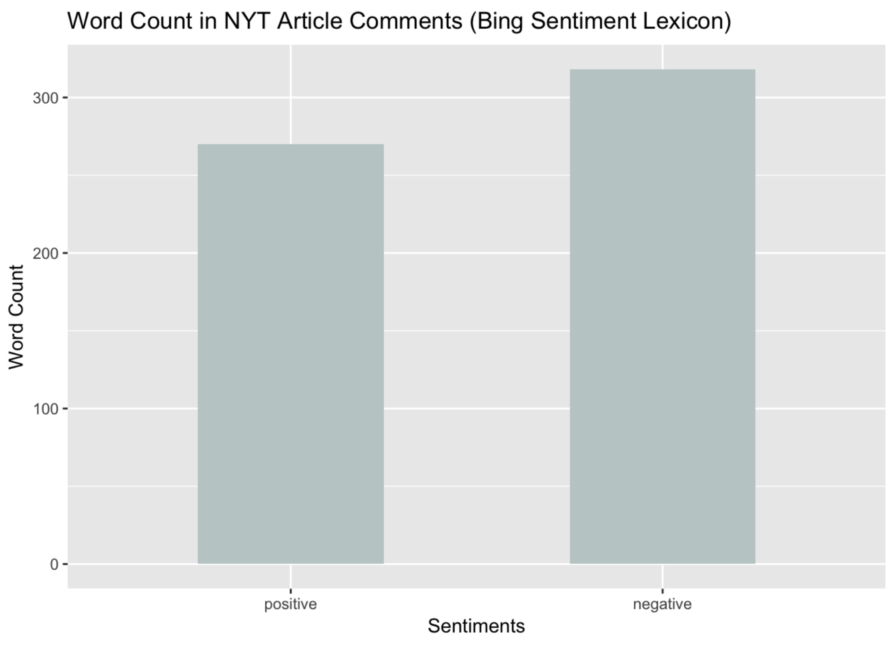
```
## Joining, by = "word"
```

```
bing_breakdown
```

```
## # A tibble: 2 x 2
##   sentiment     n
##       <chr> <int>
## 1  negative   318
## 2  positive   270
```

```
gg_bing <- ggplot(data = bing_breakdown, aes(x = reorder(sentiment, n), y = n)) +
  geom_bar(stat = 'identity', width = 0.5, fill = "azure3") +
  labs(x = "Sentiments" , y = "Word Count", title = "Word Count in NYT Article Comments (Bing Sentiment Lexicon)")


gg_bing
```



```r
# ggsave("../images/word_count_bing.pdf", gg_bing)
```

It appears that there is overall more negative sentiment than positive sentiment.

It may be useful to identify which words used in the comments are associated with each sentiment. For instance, which words most strongly invoked **negative** feelings?

```r
negative_nrc <- get_sentiments("nrc") %>%
  filter(sentiment == "negative") # extracts nrc words associated with negative sentiment

negative_words <-
  nyt_df  %>%
  inner_join(negative_nrc) %>% # inner_join will return all words from the comments that have matching nrc lexicon
words. There is no difference between using inner_join() and semi_join for our data. Each comment word can only co
unt towards one sentiment category (the negative category) since we are matching nyt_df to negative, a data frame
of nrc words that we have defined to only include words associated with negative sentiment. You will never have a
word that falls into the negative category and another category.
  count(word, sort = TRUE)
```

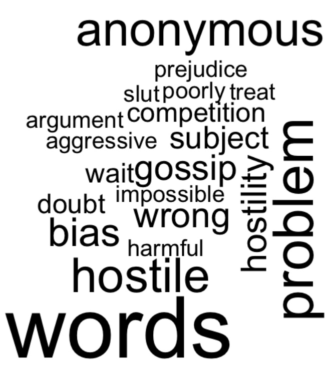
```
## Joining, by = "word"
```

```r
negative_words
```

```
## # A tibble: 184 x 2
##           word     n
##          <chr> <int>
##  1       words    18
##  2     problem    11
##  3   anonymous     9
##  4     hostile     9
##  5        bias     7
##  6      gossip     6
##  7   hostility     6
##  8       wrong     6
##  9     subject     5
## 10 competition     4
## # ... with 174 more rows
```

```r
# png("../images/wordcloud_negative")
# wordcloud(negative_words$word[1:20], negative_words$n[1:20], colors = "black")
# dev.off()

wordcloud(negative_words$word[1:20], negative_words$n[1:20], colors = "black")
```



It appears that **commenters were troubled by the anonymous nature of the posts made on Economic Job Market Rumors**. In fact, as Alice Wu's thesis explains, anonymous posting allows users to make offensive statements they don't want attached to their names. There also appears to be quite a bit of hostility surrounding this subject as the words "hostile" and "hostility" show up frequently.

To look on the brighter side, let's check out the words most associated with **positive** feelings.

```
positive_nrc <- get_sentiments("nrc") %>%
  filter(sentiment == "positive")

positive_words <-
  nyt_df  %>%
  inner_join(positive_nrc) %>%
  count(word, sort = TRUE)
```

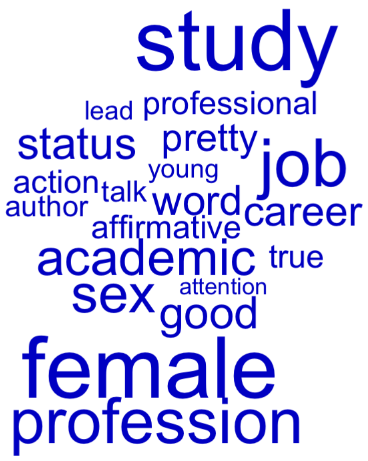
```
## Joining, by = "word"
```

```
positive_words
```

```
## # A tibble: 231 x 2
##          word     n
##         <chr> <int>
##  1     female    24
##  2      study    24
##  3        job    18
##  4 profession    17
##  5        sex    14
##  6   academic    13
##  7       good    11
##  8     status    11
##  9     career    10
## 10       word    10
## # ... with 221 more rows
```

```
# png("../images/wordcount_positive")
# wordcloud(positive_words$word[1:20], positive_words$n[1:20], colors = "medium blue")
# dev.off()

wordcloud(positive_words$word[1:20], positive_words$n[1:20], colors = "medium blue")
```



A lot of professional and career related words show up: “study”, “job”, “profession”, “academic”, and “career”. This may suggest that **maintaining objectivity and avoiding gender biases in the economics field is a conducive to creating a positive industry environment**. The word “sex” may need to be further explored by reading the comments to understand the contexts in which the word was used. In Alice Wu’s thesis, she states that commenters on the forum recognize women more for their physical features than their intellectual capabilities.

Here are the words most associated with **anger**:

```
anger_nrc <- get_sentiments("nrc") %>%
  filter(sentiment == "anger")

anger_words <-
  nyt_df  %>%
  inner_join(anger_nrc) %>%
  count(word, sort = TRUE)
```

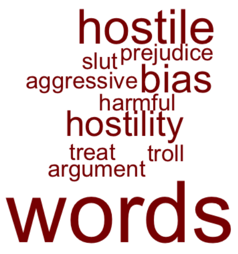
```
## Joining, by = "word"
```

```
anger_words
```

```
## # A tibble: 85 x 2
##          word     n
##         <chr> <int>
##  1      words    18
##  2    hostile     9
##  3       bias     7
##  4  hostility     6
##  5 aggressive     3
##  6   argument     3
##  7    harmful     3
##  8  prejudice     3
##  9       slut     3
## 10      treat     3
## # ... with 75 more rows
```

```
#png("../images/wordcount_anger")
#wordcloud(anger_words$word[1:20], anger_words$n[1:20], colors = "dark red")
#dev.off()

wordcloud(anger_words$word[1:20], anger_words$n[1:20], colors = "dark red")
```



Conclusion/TLDR

We've now learned the basics of text analytics and sentiment analysis, through which we can gain meaningful insight into risk management, business intelligence, smart advertising and much more across all kinds of industries.

To summarize the process, we begin by reorganizing unstructured text data into forms that are easy to work with. The data cleaning process may involve the use of multiple functions in a labor-intensive cleaning process, or it may be simplified through the usage of functions in the tidytext package. Tidytext greatly simplifies text mining through tokenization, in which tokens (words, sentences, etc.) are organized into a table, with one token per row.

After cleaning the data, we can use a simple word count to observe the frequency of certain words in the text data. By using a sentiment lexicon, we can identify the sentiments that the text data invoke by matching the data against the data frame of words and their associated sentiments provided in the lexicon. Performing sentiment analysis is useful in many contexts, one of which is the potential identification of the opinions of the creators of the text data.

As with any kind of data, visualization is key. ggplot2 is useful as always, while the wordcloud package is specifically useful to text data.

Text data analytics and sentiment analysis are growing, although they're still relatively new compared to traditional data analysis of quantitative variables. Given the growth of digital text data and the proliferation of social media, we can expect to see more usage and application of text data anlytics and sentiment analysis in the near future.

References

Data source of the 155 comments used in the example (comments were manually attracted and placed in a text file):

Evidence of a Toxic Environment for Women in Economics, New York Times

Unstructured Data and the 80 Percent Rule, Breakthrough Analysis

10 text mining examples

Getting Data from the Web with R, Gaston Sanchez

R gsub Function, End Memo

3.2 Character Classes and Bracket Expressions, GNU Operating System

Remove all punctuation except for apostrophes in R, Stack Overflow

paste, paste0, and sprintf, TRinnker's R Blog

Intro to Text Analysis with R, R-bloggers

Tidy text:
The tidy text format from Text Mining with R

Creating Tidy Text, UC Business Analytics R Programming Guide

Introduction to tidytext, CRAN