Analyzing Text: Preliminaries

Unstructured Data Analytics

```r
library(tidyverse)
library(stringr)
```

1. Project Gutenberg

The **gutenbergr** package provides several functions that allow us to import public domain books from Project Gutenberg (www.gutenberg.org).

```r
library(gutenbergr)
```

In order to import a book, we first need the book id. However, if we do not know the book id, we can search for it by title. The `gutenberg_works()` function provides us with metadata about each of the works in the library. For example, we can get metadata about "The War of the Worlds":

```r
gutenberg_works() %>%
  filter(title == "The War of the Worlds") %>%
  glimpse()
```

```
## Rows: 1
## Columns: 8
## $ gutenberg_id        <int> 36
## $ title               <chr> "The War of the Worlds"
## $ author              <chr> "Wells, H. G. (Herbert George)"
## $ gutenberg_author_id <int> 30
## $ language            <chr> "en"
## $ gutenberg_bookshelf <chr> "Movie Books/Science Fiction"
## $ rights              <chr> "Public domain in the USA."
## $ has_text            <lgl> TRUE
```

From the metadata, we get the book id, title, author, etc. Using the book id, we can now download the book by passing the id to the `gutenberg_download()` function:

```r
#if you need to find a mirror, run this first
#gutenberg_get_mirror()
war_worlds <- gutenberg_download(36, mirror = 'http://aleph.gutenberg.org')
slice(war_worlds, 1:300)
```

```
## # A tibble: 300 x 2
##    gutenberg_id text
##           <int> <chr>
##   1          36 "cover "
##   2             36 ""
##   3             36 ""
##   4             36 ""
##   5             36 ""
##   6             36 "The War of the Worlds"
##   7             36 ""
##   8             36 "by H. G. Wells"
##   9             36 ""
##  10             36 ""
## # ... with 290 more rows
```

Here we see the first 300 lines of the work. It seems this particular work is broken down into books and chapters. In order to analyze the work at the line, chapter and book levels, we could add new features to our data to represent these values.

Adding new features isn't "text pre-processing" but it is a kind of data preparation that adds useful analytical structures for later

Regular expressions are handy adding features (and other types of pre-processing), which is one reason we'll look into them:

```
war_worlds <- war_worlds %>%
  mutate(
    linenumber = row_number(),
    chapter = cumsum(str_detect(
      text, regex("^[IVX]*\\.$", ignore_case = FALSE)
    )),
    book = cumsum(str_detect(
      text, regex("^BOOK [\\w]", ignore_case = FALSE)
    ))
  )
slice(war_worlds, 1:300)
```

```
## # A tibble: 300 x 5
##    gutenberg_id text                         linenumber chapter  book
##           <int> <chr>                             <int>   <int> <int>
##   1             36 "cover "                           1       0     0
##   2             36 ""                                 2       0     0
##   3             36 ""                                 3       0     0
##   4             36 ""                                 4       0     0
##   5             36 ""                                 5       0     0
##   6             36 "The War of the Worlds"            6       0     0
##   7             36 ""                                 7       0     0
##   8             36 "by H. G. Wells"                   8       0     0
##   9             36 ""                                 9       0     0
##  10             36 ""                                10       0     0
## # ... with 290 more rows
```

Question: Can you explain what is happening here?

The `gutenberg_download()` function does allow us to download more than one book at a time. For example, we can download books *768* and *1260* at the same time:

```
books <- gutenberg_download(c(768, 1260),
                            meta_fields = "title",
                            mirror = 'https://gutenberg.pglaf.org/')
slice(books, 1:10)
```

```
## # A tibble: 10 x 3
##    gutenberg_id text                 title
##           <int> <chr>                <chr>
##  1          768 "Wuthering Heights"  Wuthering Heights
##  2          768 ""                   Wuthering Heights
##  3          768 "by Emily Brontë"    Wuthering Heights
##  4          768 ""                   Wuthering Heights
##  5          768 ""                   Wuthering Heights
##  6          768 ""                   Wuthering Heights
##  7          768 ""                   Wuthering Heights
##  8          768 "CHAPTER I"          Wuthering Heights
##  9          768 ""                   Wuthering Heights
## 10          768 ""                   Wuthering Heights
```

Note that this time, we also included an additional argument (`meta_fields = "title"`). This is so that the title of each book is included as a column in the data. Therefore, we can find out what the distinct book titles are:

```
books %>%
  distinct(title)
```

```
## # A tibble: 2 x 1
##   title
##   <chr>
## 1 Wuthering Heights
## 2 Jane Eyre: An Autobiography
```

In the previous examples, we searched for books based on the title and/or the book id. What if we don't know the exact title of a book or the book id? In such a case, we again make use of regular expressions and the `string_detect()` functions from the `stringr` package. For example, we could search for works where the author matches "wells" and the book title matches "war":

```
gutenberg_works() %>%
  filter(str_detect(str_to_lower(author),"wells")) %>%
  filter(str_detect(str_to_lower(title), "war")) %>%
  select(gutenberg_id, title, author)
```

```
## # A tibble: 8 x 3
##   gutenberg_id title                                                     author
##          <int> <chr>                                                     <chr>
## 1           36 "The War of the Worlds"                                   Wells~
## 2          780 "The War in the Air"                                      Wells~
## 3         1804 "War and the Future: Italy, France and Britain at War"    Wells~
## 4         3383 "Spanish Prisoners of War (from Literature and Life)"     Howel~
## 5         3690 "Floor Games; a companion volume to \"Little Wars\""      Wells~
## 6         3691 "Little Wars; a game for boys from twelve years of age to~ Wells~
## 7         8386 "Ptomaine Street: The Tale of Warble Petticoat"           Wells~
## 8        11289 "What is Coming? A Forecast of Things after the War"      Wells~
```

As you can imagine, we can get very creative with our regular expressions in order to match exactly what we want.

Besides the `gutenberg_works()` function which returns metadata about a work, other commonly used meta-datasets from the `gutenbergr` package include:

1. `gutenberg_subjects` We use this dataset to enumerate books from a particular topic or genre:

```
gutenberg_subjects %>%
  filter(subject == "Science fiction") %>%
  slice(1:10)
```

```
## # A tibble: 10 x 3
##    gutenberg_id subject_type subject
##           <int> <chr>        <chr>
##  1           35 lcsh         Science fiction
##  2           36 lcsh         Science fiction
##  3           42 lcsh         Science fiction
##  4           43 lcsh         Science fiction
##  5           62 lcsh         Science fiction
##  6           64 lcsh         Science fiction
##  7           68 lcsh         Science fiction
##  8           72 lcsh         Science fiction
##  9           83 lcsh         Science fiction
## 10           84 lcsh         Science fiction
```

2. `gutenberg_authors` We use this dataset to get information about the author of a work:

```
gutenberg_authors %>%
  filter(author == "Austen, Jane") %>%
  glimpse()
```

```
## Rows: 1
## Columns: 7
## $ gutenberg_author_id <int> 68
## $ author              <chr> "Austen, Jane"
## $ alias               <chr> NA
## $ birthdate           <int> 1775
## $ deathdate           <int> 1817
## $ wikipedia           <chr> "http://en.wikipedia.org/wiki/Jane_Austen"
## $ aliases             <chr> NA
```

Pre-processing and first explorations

2. Tokenization

Before we can analyze unstructured text data, we often need to transform it into structured form through a series of pre-processing steps. One of those steps is tokenization. To illustrate how to tokenize text, let's start with our "favorite" nursery rhyme:

```r
rhyme <-
  c(
    "Hey, diddle, diddle,",
    "The cat and the fiddle,",
    "The cow jumped over the moon;",
    "The little dog laughed",
    "To see such sport,",
    "And the dish ran away with the spoon."
  )

rhyme
```

```
## [1] "Hey, diddle, diddle,"
## [2] "The cat and the fiddle,"
## [3] "The cow jumped over the moon;"
## [4] "The little dog laughed"
## [5] "To see such sport,"
## [6] "And the dish ran away with the spoon."
```

Before we tokenize the rhyme, let's convert it to a tibble and add a number for each line:

```r
rhyme <- tibble(line=1:6, text = rhyme)

rhyme
```

```
## # A tibble: 6 x 2
##    line text
##   <int> <chr>
## 1     1 Hey, diddle, diddle,
## 2     2 The cat and the fiddle,
## 3     3 The cow jumped over the moon;
## 4     4 The little dog laughed
## 5     5 To see such sport,
## 6     6 And the dish ran away with the spoon.
```

Now we are ready to tokenize our text. To do this, we make use of the `unnest_tokens()` function from the `tidytext` package.

```r
library(tidytext)
rhyme %>%
  unnest_tokens(output = word, input = text, token = "words")
```

```
## # A tibble: 30 x 2
##     line word
##    <int> <chr>
##  1     1 hey
##  2     1 diddle
##  3     1 diddle
##  4     2 the
##  5     2 cat
##  6     2 and
##  7      2 the
##  8      2 fiddle
##  9      3 the
## 10      3 cow
## # ... with 20 more rows
```

Notice that we set `token = "words"`. This specifies the granularity at which we intend to tokenize (words). We could also set `token = "ngrams"` and `n = 2`. This means that we intend to tokenize with bigrams.

```
rhyme %>%
  unnest_tokens(output = word, input = text, token = "ngrams", n = 2)
```

```
## # A tibble: 24 x 2
##     line word
##    <int> <chr>
##  1      1 hey diddle
##  2      1 diddle diddle
##  3      2 the cat
##  4      2 cat and
##  5      2 and the
##  6      2 the fiddle
##  7      3 the cow
##  8      3 cow jumped
##  9      3 jumped over
## 10      3 over the
## # ... with 14 more rows
```

What if we want to use whole sentences as tokens? Easy. We set `token = "sentences"`.

```
rhyme %>%
  unnest_tokens(output = word, input = text, token = "sentences")
```

```
## # A tibble: 6 x 2
##    line word
##   <int> <chr>
## 1     1 hey, diddle, diddle,
## 2     2 the cat and the fiddle,
## 3     3 the cow jumped over the moon;
## 4     4 the little dog laughed
## 5     5 to see such sport,
## 6     6 and the dish ran away with the spoon.
```

After we tokenize we can answer simple questions like "how many words?"

```
rhyme %>%
  unnest_tokens(output = word, input = text, token = "words") %>%
  nrow()
```

```
## [1] 30
```

Or questions like "how many words per line?"

```
rhyme %>%
  unnest_tokens(output = word, input = text, token = "words") %>%
  group_by(line) %>%
  summarize(wordcount = n())
```

```
## # A tibble: 6 x 2
##    line wordcount
##   <int>     <int>
## 1     1         3
## 2     2         5
## 3     3         6
## 4     4         4
## 5     5         4
## 6     6         8
```

Counting words can be useful if we care about volume of text. For example, maybe our poems aren't successful and we're worried they are too long for a nursery rhyme. We could check various poems for lengths and see how we fall in the distribution. Maybe we think that longer poems are predictive of poetry books making more money, so we want average poem-length to add to a regression analysis. As soon as you have numbers, you can start doing numbery-things.

Another approach looks at frequency of words, in which case it's often helpful to remove words that are very common in the language but don't provide a lot of analytical value when trying to understand the content of the text

3. Stop Words

The `tidytext` package provides a dataset of common stop words. We can use this dataset to remove stop words from our data by using an `anti_join()`. To do this, we first need to import the stop words dictionary into our environment:

```
data("stop_words")
stop_words
```

```
## # A tibble: 1,149 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
##  7 accordingly SMART
##  8 across      SMART
##  9 actually    SMART
## 10 after       SMART
## # ... with 1,139 more rows
```

Then we remove the stop words from our text by using the `anti_join()` function:

```
rhyme %>%
  unnest_tokens(output = word, input = text, token = "words") %>%
  anti_join(stop_words, by = "word")
```

```
## # A tibble: 14 x 2
##     line word
##    <int> <chr>
##  1     1 hey
##  2     1 diddle
##  3     1 diddle
##  4     2 cat
##  5     2 fiddle
##  6     3 cow
##  7     3 jumped
##  8     3 moon
##  9     4 dog
## 10     4 laughed
## 11     5 sport
## 12     6 dish
## 13     6 ran
## 14     6 spoon
```

Notice that words like “the” and “and” are now gone.

4. Custom Stop Words

The `tidytext` stop words come from three different lexicons (or dictionaries) - `onix`, `SMART` and `snowball`.

```
stop_words
```

```
## # A tibble: 1,149 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
##  7 accordingly SMART
##  8 across      SMART
##  9 actually    SMART
## 10 after       SMART
## # ... with 1,139 more rows
```

We can get the number of words from each lexicon:

```
stop_words %>%
  count(lexicon)
```

```
## # A tibble: 3 x 2
##   lexicon      n
##   <chr>    <int>
## 1 onix       404
## 2 SMART      571
## 3 snowball   174
```

We can also create our own list of stop words. For example, let's assume that we want to treat the words "diddle" and "fiddle" as stop words. We start by creating a new stop words dictionary with those two words:

```
new_stop_words <- tibble(
  word = c("diddle", "fiddle"),
  lexicon = c("custom")
)
new_stop_words
```

```
## # A tibble: 2 x 2
##   word   lexicon
##   <chr>  <chr>
## 1 diddle custom
## 2 fiddle custom
```

Then we combine the new dictionary with the existing one to create a new dictionary of stop words called `custom_stop_words`:

```
custom_stop_words <-
  bind_rows(new_stop_words, stop_words)

custom_stop_words
```

```
## # A tibble: 1,151 x 2
##    word        lexicon
##    <chr>       <chr>
##  1 diddle      custom
##  2 fiddle      custom
##  3 a           SMART
##  4 a's         SMART
##  5 able        SMART
##  6 about       SMART
##  7 above       SMART
##  8 according   SMART
##  9 accordingly SMART
## 10 across      SMART
## # ... with 1,141 more rows
```

We can get a count of the number of words in each lexicon:

```
custom_stop_words %>%
  count(lexicon)
```

```
## # A tibble: 4 x 2
##   lexicon      n
##   <chr>    <int>
## 1 custom       2
## 2 onix       404
## 3 SMART      571
## 4 snowball   174
```

And just like we did with the default dictionary, we can use our custom dictionary to remove stop words in our text:

```
rhyme %>%
  unnest_tokens(output = word, input = text, token = "words") %>%
  anti_join(custom_stop_words, by = "word")
```

```
## # A tibble: 11 x 2
##     line word
##    <int> <chr>
##  1     1 hey
##  2     2 cat
##  3     3 cow
##  4     3 jumped
##  5     3 moon
##  6     4 dog
##  7     4 laughed
##  8     5 sport
##  9     6 dish
## 10     6 ran
## 11     6 spoon
```