Text Mining (Natural Language Processing)

JSC 370: Data Science II

Natural Language Processing (NLP) is used for qualitative data that is collected using open ended or free form text from a survey, medical provider notes in an electronic medical record (EMR), or a transcript of research participant interviews (Koleck et al., 2019).

It is also called 'text mining'.

What is NLP used for?

- Looking at frequencies of words and phrases in text.
- Labeling relationships between words such as subject, object, modification.
- Identify entities in free text, labeling them with types such as person,

location, organization.

- Coupled with AI it can predict words (autocomplete).

How can we do NLP?

- We turn text into numbers.
- Then use R and the tidyverse to explore those numbers.

tidytext: Text mining using dplyr, ggplot2, and other tidy tools

Authors: Julia Silge, David Robinson
License: MIT

R-CMD-check failing | build passing | build passing | CRAN 0.2.3 | CRAN Not OK | coverage 84%
DOI 10.5281/zenodo.3697085 | JOSS 10.21105/joss.00037 | downloads 35K/month | downloads 928K

Using tidy data principles can make many text mining tasks easier, more effective, and consistent with tools already in wide use. Much of the infrastructure needed for text mining with tidy data frames already exists in packages like dplyr, broom, tidyr and ggplot2. In this package, we provide functions and supporting data sets to allow conversion of text to and from tidy formats, and to switch seamlessly between tidy tools and existing text mining packages. Check out our book to learn more about text mining using tidy data principles.

Why tidytext?

Works seemlessly with ggplot2, dplyr and tidyr.

Alternatives:

R: quanteda, tm, koRpus

Python: nltk, Spacy, gensim

Alice's Adventures in Wonderland

Download the alice dataset from here. There are 12 chapters

```r
library(tidyverse)
alice <- readRDS("alice.rds")
alice
```

```
## # A tibble: 3,351 × 3
##    text                                                        chapter chapter_name
##    <chr>                                                         <int> <chr>
##  1 "CHAPTER I."                                                      1 CHAPTER I.
##  2 "Down the Rabbit-Hole"                                            1 CHAPTER I.
##  3 ""                                                                1 CHAPTER I.
##  4 ""                                                                1 CHAPTER I.
##  5 "Alice was beginning to get very tired of sitting by he…          1 CHAPTER I.
##  6 "bank, and of having nothing to do: once or twice she h…          1 CHAPTER I.
##  7 "the book her sister was reading, but it had no picture…          1 CHAPTER I.
##  8 "conversations in it, “and what is the use of a book,” …          1 CHAPTER I.
##  9 "“without pictures or conversations?”"                            1 CHAPTER I.
## 10 ""                                                                1 CHAPTER I.
```

Tokenizing

Turning text into smaller units, essentially splitting a sentence, phrase, paragraph or entire document into smaller units called tokens (i.e. individual words, numbers, or punctuation marks). Tokenization is needed for natural language processing.

In English:

- split by spaces
- more advanced algorithms

Spacy tokenizer

1. Iterate over whitespace-separated substrings.
2. Look for a token match. If there is a match, stop processing and keep this token.
3. Check whether we have an explicitly defined special case for this substring. If we do, use it.
4. Otherwise, try to consume one prefix. If we consumed a prefix, go back to #2, so that the token match and special cases always get priority.
5. If we didn’t consume a prefix, try to consume a suffix and then go back to #2.
6. If we can’t consume a prefix or a suffix, look for a URL match.
7. If there’s no URL match, then look for a special case.
8. Look for “infixes” — stuff like hyphens etc. and split the substring into tokens on all infixes.
9. Once we can’t consume any more of the string, handle it as a single token.

Tokenizing with unnest_tokens

```
library(tidytext)
alice %>%
  unnest_tokens(token, text)
```

```
## # A tibble: 26,687 × 3
##    chapter chapter_name token
##      <int> <chr>        <chr>
##  1       1 CHAPTER I.   chapter
##  2       1 CHAPTER I.   i
##  3       1 CHAPTER I.   down
##  4       1 CHAPTER I.   the
##  5       1 CHAPTER I.   rabbit
##  6       1 CHAPTER I.   hole
##  7       1 CHAPTER I.   alice
##  8       1 CHAPTER I.   was
##  9       1 CHAPTER I.   beginning
## 10       1 CHAPTER I.   to
## # … with 26,677 more rows
```

Words as a unit

Now that we have words as the observation unit we can use the dplyr toolbox.

Using dplyr verbs

```
library(dplyr)
alice %>%
  unnest_tokens(token, text)
```

```
## # A tibble: 26,687 × 3
##    chapter chapter_name token
##      <int> <chr>        <chr>
##  1       1 CHAPTER I.   chapter
##  2       1 CHAPTER I.   i
##  3       1 CHAPTER I.   down
##  4       1 CHAPTER I.   the
##  5       1 CHAPTER I.   rabbit
##  6       1 CHAPTER I.   hole
##  7       1 CHAPTER I.   alice
##  8       1 CHAPTER I.   was
##  9       1 CHAPTER I.   beginning
## 10       1 CHAPTER I.   to
## # … with 26,677 more rows
```

Using dplyr verbs

```
library(dplyr)
alice %>%
  unnest_tokens(token, text) %>%
  count(token)
```

```
## # A tibble: 2,740 × 2
##    token        n
##    <chr>    <int>
##  1 _alice's     1
##  2 _all         1
##  3 _all_        1
##  4 _and         1
##  5 _are_        4
##  6 _at          1
##  7 _before      1
##  8 _beg_        1
##  9 _began_      1
## 10 _best_       2
## # … with 2,730 more rows
```

Using dplyr verbs

```
library(dplyr)
alice %>%
  unnest_tokens(token, text) %>%
  count(token, sort = TRUE)
```

```
## # A tibble: 2,740 × 2
##    token     n
##    <chr> <int>
##  1 the    1643
##  2 and     871
##  3 to      729
##  4 a       632
##  5 she     538
##  6 it      527
##  7 of      514
##  8 said    460
##  9 i       393
## 10 alice   386
## # … with 2,730 more rows
```

Using dplyr verbs

```
library(dplyr)
alice %>%
  unnest_tokens(token, text) %>%
  count(chapter, token)
```

```
## # A tibble: 7,549 × 3
##    chapter token            n
##      <int> <chr>        <int>
##  1       1 _curtseying_     1
##  2       1 _never_          1
##  3       1 _not_            1
##  4       1 _one_            1
##  5       1 _poison_         1
##  6       1 _that_           1
##  7       1 _through_        1
##  8       1 _took            1
##  9       1 _very_           4
## 10       1 _was_            1
## # … with 7,539 more rows
```

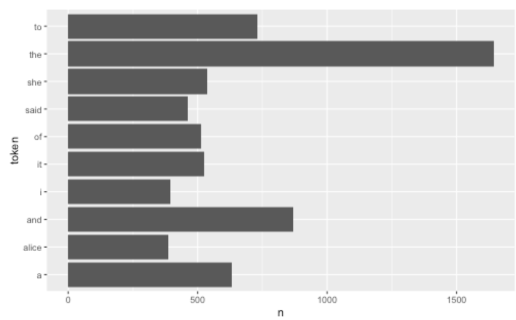
Using dplyr verbs

```
library(dplyr)
alice %>%
  unnest_tokens(token, text) %>%
  group_by(chapter) %>%
  count(token) %>%
  top_n(10, n)
```

```
## # A tibble: 122 x 3
## # Groups:   chapter [12]
##    chapter token     n
##      <int> <chr> <int>
##  1       1 a        52
##  2       1 alice    27
##  3       1 and      65
##  4       1 i        30
##  5       1 it       62
##  6       1 of       43
##  7       1 she      79
##  8       1 the      92
##  9       1 to       75
```

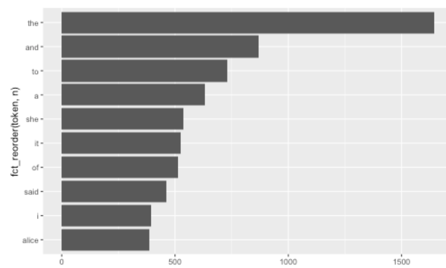
Using dplyr verbs and ggplot2

```r
library(dplyr)
library(ggplot2)
alice %>%
  unnest_tokens(token, text) %>%
  count(token) %>%
  top_n(10, n) %>%
  ggplot(aes(n, token)) +
  geom_col()
```



Using dplyr verbs and ggplot2

```
library(dplyr)
library(ggplot2)
library(forcats)
alice %>%
  unnest_tokens(token, text) %>%
  count(token) %>%
  top_n(10, n) %>%
  ggplot(aes(n, fct_reorder(token, n))) +
  geom_col()
```



Stop words

A lot of the words don't tell us very much. Words such as "the", "and", "at" and "for" appear a lot in English text but doesn't add much to the context.

Words such as these are called stop words

For more information about differences in stop words and when to remove them read this chapter https://smltar.com/stopwords

Stop words in tidytext

tidytext comes with a data.frame of stop words

```
stop_words
```

```
## # A tibble: 1,149 × 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
```

snowball stopwords

```
##   [1] "i"          "me"         "my"         "myself"     "we"
##   [6] "our"        "ours"       "ourselves"  "you"        "your"
##  [11] "yours"      "yourself"   "yourselves" "he"         "him"
##  [16] "his"        "himself"    "she"        "her"        "hers"
##  [21] "herself"    "it"         "its"        "itself"     "they"
##  [26] "them"       "their"      "theirs"     "themselves" "what"
##  [31] "which"      "who"        "whom"       "this"       "that"
##  [36] "these"      "those"      "am"         "is"         "are"
##  [41] "was"        "were"       "be"         "been"       "being"
##  [46] "have"       "has"        "had"        "having"     "do"
##  [51] "does"       "did"        "doing"      "would"      "should"
##  [56] "could"      "ought"      "i'm"        "you're"     "he's"
##  [61] "she's"      "it's"       "we're"      "they're"    "i've"
##  [66] "you've"     "we've"      "they've"    "i'd"        "you'd"
##  [71] "he'd"       "she'd"      "we'd"       "they'd"     "i'll"
##  [76] "you'll"     "he'll"      "she'll"     "we'll"      "they'll"
##  [81] "isn't"      "aren't"     "wasn't"     "weren't"    "hasn't"
##  [86] "haven't"    "hadn't"     "doesn't"    "don't"      "didn't"
##  [91] "won't"      "wouldn't"   "shan't"     "shouldn't"  "can't"
##  [96] "cannot"     "couldn't"   "mustn't"    "let's"      "that's"
```

Removing stopwords

We can use an `anti_join()` to remove the tokens that also appear in the `stop_words` data.frame

```
alice %>%
  unnest_tokens(token, text) %>%
  anti_join(stop_words, by = c("token" = "word")) %>%
  count(token, sort = TRUE)
```

```
## # A tibble: 2,314 × 2
##    token       n
##    <chr>   <int>
##  1 alice     386
##  2 time       71
##  3 queen      68
##  4 king       61
##  5 don’t      60
##  6 it’s       57
##  7 i’m        56
##  8 mock       56
```

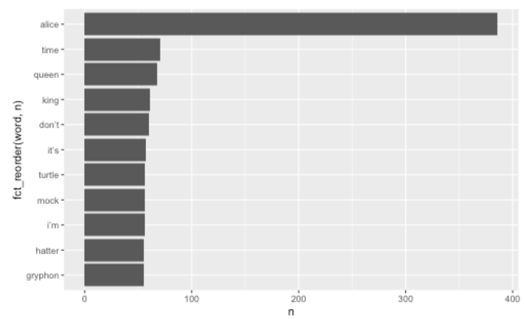
Anti-join with same variable name

```
alice %>%
  unnest_tokens(word, text) %>%
  anti_join(stop_words, by = c("word")) %>%
  count(word, sort = TRUE)
```

```
## # A tibble: 2,314 × 2
##    word        n
##    <chr>   <int>
##  1 alice     386
##  2 time       71
##  3 queen      68
##  4 king       61
##  5 don’t      60
##  6 it’s       57
##  7 i’m        56
##  8 mock       56
##  9 turtle     56
## 10 gryphon    55
## # … with 2,304 more rows
```

Stop words removed

```
alice %>%
  unnest_tokens(word, text) %>%
  anti_join(stop_words, by = c("word")) %>%
  count(word, sort = TRUE) %>%
  top_n(10, n) %>%
  ggplot(aes(n, fct_reorder(word, n))) +
  geom_col()
```



Wordcloud

```
library(wordcloud)
pal<-brewer.pal(8,"Spectral")
alice %>%
  unnest_tokens(word, text) %>%
  anti_join(stop_words, by = c("word")) %>%
  count(word, sort = TRUE) %>%
  top_n(10, n) %>%
  with(wordcloud(word, n, random.order = FALSE, max.words = 100, colors=pal))
```

Which words appear together?

ngrams are n consecutive word, we can count these to see what words appears together.

- ngram with n = 1 are called unigrams: "which", "words", "appears", "together"
- ngram with n = 2 are called bigrams: "which words", "words appears", "appears together"
- ngram with n = 3 are called trigrams: "which words appears", "words appears together"

Which words appears together?

We can extract bigrams using `unnest_ngrams()` with `n = 2`

```
alice %>%
  unnest_ngrams(ngram, text, n = 2)
```

```
## # A tibble: 25,170 × 3
##    chapter chapter_name ngram
##      <int> <chr>        <chr>
##  1       1 CHAPTER I.   chapter i
##  2       1 CHAPTER I.   down the
##  3       1 CHAPTER I.   the rabbit
##  4       1 CHAPTER I.   rabbit hole
##  5       1 CHAPTER I.   <NA>
##  6       1 CHAPTER I.   <NA>
##  7       1 CHAPTER I.   alice was
##  8       1 CHAPTER I.   was beginning
##  9       1 CHAPTER I.   beginning to
## 10       1 CHAPTER I.   to get
## # … with 25,160 more rows
```

Which words appears together?

Tallying up the bi-grams still shows a lot of stop words but is able to pick up relationships with patients

```
alice %>%
  unnest_ngrams(ngram, text, n = 2) %>%
  count(ngram, sort = TRUE)
```

```
## # A tibble: 13,424 x 2
##    ngram          n
##    <chr>      <int>
##  1 <NA>         951
##  2 said the     206
##  3 of the       130
##  4 said alice   112
##  5 in a          96
##  6 and the       75
##  7 in the        75
##  8 it was        72
```

Which words appears together?

```
alice %>%
  unnest_ngrams(ngram, text, n = 2) %>%
  separate(ngram, into = c("word1", "word2"), sep = " ") %>%
  select(word1, word2)
```

```
## # A tibble: 25,170 × 2
##    word1     word2
##    <chr>     <chr>
##  1 chapter   i
##  2 down      the
##  3 the       rabbit
##  4 rabbit    hole
##  5 <NA>      <NA>
##  6 <NA>      <NA>
##  7 alice     was
##  8 was       beginning
##  9 beginning to
## 10 to        get
## # … with 25,160 more rows
```

```r
alice %>%
  unnest_ngrams(ngram, text, n = 2) %>%
  separate(ngram, into = c("word1", "word2"), sep = " ") %>%
  select(word1, word2) %>%
  filter(word1 == "alice")
```

```
## # A tibble: 336 × 2
##    word1 word2
##    <chr> <chr>
##  1 alice was
##  2 alice think
##  3 alice started
##  4 alice after
##  5 alice had
##  6 alice to
##  7 alice had
##  8 alice had
##  9 alice soon
## 10 alice began
## # … with 326 more rows
```

```r
alice %>%
  unnest_ngrams(ngram, text, n = 2) %>%
  separate(ngram, into = c("word1", "word2"), sep = " ") %>%
  select(word1, word2) %>%
  filter(word1 == "alice") %>%
  count(word2, sort = TRUE)
```

```
## # A tibble: 133 × 2
##    word2       n
##    <chr>   <int>
##  1 and        18
##  2 was        17
##  3 thought    12
##  4 as         11
##  5 said       11
##  6 could      10
##  7 had        10
##  8 did         9
##  9 in          9
## 10 to          9
## # … with 123 more rows
```

```
alice %>%
  unnest_ngrams(ngram, text, n = 2) %>%
  separate(ngram, into = c("word1", "word2"), sep = " ") %>%
  select(word1, word2) %>%
  filter(word2 == "alice") %>%
  count(word1, sort = TRUE)
```

```
## # A tibble: 106 × 2
##    word1         n
##    <chr>     <int>
##  1 said        112
##  2 thought      25
##  3 to           22
##  4 and          15
##  5 poor         11
##  6 cried         7
##  7 at            6
##  8 so            6
##  9 that          5
## 10 exclaimed     3
## # … with 96 more rows
```

TF-IDF

TF: Term frequency gives weight to terms that appear a lot. It's a measure of how important a word may be and how frequently a word occurs within a document (e.g. a book chapter). IDF decreases the weight for commonly used words and increases the weight for words that are not used very much in a collection of documents (e.g. all chapters in a book).

Some words that occur many times in a document may not be important; in English, these are probably words like “the”, “is”, “of”, and so forth. We might take the approach of adding words like these to a list of stop words and removing them before analysis, but it is possible that some of these words might be more important in some documents than others. A list of stop words is not a sophisticated approach to adjusting term frequency for commonly used words.

TF-IDF

IDF: Inverse document frequency

IDF decreases the weight for commonly used words and increases the weight for words that are not used very much in a collection of documents.

The inverse document frequency for any given term is defined as

$$idf(term) = ln(\frac{\text{n documents}}{\text{n documents containing term}})$$

TF-IDF

TF-IDF: TF and IDF can be combined (the two quantities multiplied together), which is the frequency of a term adjusted for how rarely it is used.

The idea of TF-IDF is to find the important words for the content of each document by decreasing the weight for commonly used words and increasing the weight for words that are not used very much in a collection or corpus of documents.

TF-IDF with tidytext

```
alice %>%
  unnest_tokens(text, text)
```

```
## # A tibble: 26,687 × 3
##    text      chapter chapter_name
##    <chr>       <int> <chr>
##  1 chapter         1 CHAPTER I.
##  2 i               1 CHAPTER I.
##  3 down            1 CHAPTER I.
##  4 the             1 CHAPTER I.
##  5 rabbit          1 CHAPTER I.
##  6 hole            1 CHAPTER I.
##  7 alice           1 CHAPTER I.
##  8 was             1 CHAPTER I.
##  9 beginning       1 CHAPTER I.
## 10 to              1 CHAPTER I.
## # … with 26,677 more rows
```

TF-IDF with tidytext

```
alice %>%
  unnest_tokens(text, text) %>%
  count(text, chapter)
```

```
## # A tibble: 7,549 × 3
##    text      chapter     n
##    <chr>       <int> <int>
##  1 _alice's        2     1
##  2 _all           12     1
##  3 _all_          12     1
##  4 _and            9     1
##  5 _are_           4     1
##  6 _are_           6     1
##  7 _are_           8     1
##  8 _are_           9     1
##  9 _at             9     1
## 10 _before        12     1
## # … with 7,539 more rows
```

TF-IDF with tidytext

```
alice %>%
  unnest_tokens(text, text) %>%
  count(text, chapter) %>%
  bind_tf_idf(text, chapter, n)
```

```
## # A tibble: 7,549 × 6
##    text     chapter     n       tf   idf   tf_idf
##    <chr>      <int> <int>    <dbl> <dbl>    <dbl>
##  1 _alice's       2     1 0.000471  2.48 0.00117
##  2 _all          12     1 0.000468  2.48 0.00116
##  3 _all_         12     1 0.000468  2.48 0.00116
##  4 _and           9     1 0.000435  2.48 0.00108
##  5 _are_          4     1 0.000375  1.10 0.000411
##  6 _are_          6     1 0.000382  1.10 0.000420
##  7 _are_          8     1 0.000400  1.10 0.000439
##  8 _are_          9     1 0.000435  1.10 0.000478
##  9 _at            9     1 0.000435  2.48 0.00108
## 10 _before       12     1 0.000468  2.48 0.00116
## # … with 7,539 more rows
```

TF-IDF with tidytext

```
alice %>%
  unnest_tokens(text, text) %>%
  count(text, chapter) %>%
  bind_tf_idf(text, chapter, n) %>%
  arrange(desc(tf_idf))
```

```
## # A tibble: 7,549 × 6
##    text        chapter     n      tf   idf tf_idf
##    <chr>         <int> <int>   <dbl> <dbl>  <dbl>
##  1 dormouse          7    26 0.0112   1.79 0.0201
##  2 hatter            7    32 0.0138   1.39 0.0191
##  3 mock             10    28 0.0136   1.39 0.0189
##  4 turtle           10    28 0.0136   1.39 0.0189
##  5 gryphon          10    31 0.0151   1.10 0.0166
##  6 turtle            9    27 0.0117   1.39 0.0163
##  7 caterpillar       5    25 0.0115   1.39 0.0159
##  8 dance            10    13 0.00632  2.48 0.0157
##  9 mock              9    26 0.0113   1.39 0.0157
## 10 hatter           11    21 0.0110   1.39 0.0153
```

Sentiment Analysis

- Sentiment Analysis is a process of extracting opinions that have different scores like positive, negative or neutral.
- Based on sentiment analysis, you can find out the nature of opinion or sentences in text.
- Sentiment Analysis is a type of classification where the data are classified into different classes like positive or negative or happy, sad, angry, etc.

Sentiment Analysis

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

alice %>%
  unnest_tokens(word, text) %>%
  anti_join(stop_words, by = c("word")) %>%
    semi_join(positive) %>%
    count(word, sort = TRUE)
```

```
## # A tibble: 140 x 2
##    word          n
##    <chr>     <int>
##  1 beautiful    13
##  2 majesty      12
##  3 glad         11
##  4 bright        8
##  5 eagerly       8
##  6 ready         8
##  7 top           8
```

Sentiment Analysis

```r
bing <- get_sentiments("bing")
alicesentiment<-alice %>%
  unnest_tokens(word, text) %>%
  anti_join(stop_words, by = c("word")) %>%
  inner_join(bing) %>%
  count(word, sentiment, sort = TRUE)
alicesentiment
```

```
## # A tibble: 413 × 3
##    word      sentiment     n
##    <chr>     <chr>     <int>
##  1 mock      negative     56
##  2 poor      negative     27
##  3 hastily   negative     16
##  4 mad       negative     15
##  5 anxiously negative     14
##  6 beautiful positive     13
##  7 afraid    negative     12
##  8 majesty   positive     12
```

Sentiment Analysis

```
alicesentiment %>%
  filter(n > 7) %>%
  mutate(n = ifelse(sentiment == "negative", -n, n)) %>%
  mutate(word = reorder(word, n)) %>%
  ggplot(aes(word, n, fill = sentiment)) +
  geom_col() +
  coord_flip() +
  labs(y = "Contribution to sentiment")
```