Text Mining (
Language Pro

JSC 370: Data Sc

February 26, 2

What is NLP?

Natural Language Processing (NLP) is used fo
collected using open ended or free form text
provider notes in an electronic medical recor
research participant interviews (Koleck et al.,

It is also called 'text mining'.

What is NLP used for?

- Looking at frequencies of words and phr

- Labeling relationships between words su
modification.

- Identify entities in free text, labeling the
location, organization.

- Coupled with AI it can predict words (au

How can we do NLP?

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

tidytext: Text mining using dplyr, ggplo

Why tidytext?

Works seemlessly with ggplot2, dplyr and tid

Alternatives:

R: quanteda, tm, koRpus

Python: nltk, Spacy, gensim

Alice's Adventures in W

Download the alice dataset from here. There

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

```
## # A tibble: 3,351 × 3
##    text
##    <chr>
##  1 "CHAPTER I."
##  2 "Down the Rabbit-Hole"
##  3 ""
##  4 ""
##  5 "Alice was beginning to get very tired of sitting by
##  6 "bank, and of having nothing to do: once or twice sh
```

```
##  7 "the book her sister was reading, but it had no pict
##  8 "conversations in it, “and what is the use of a book
##  9 "“without pictures or conversations?”"
```

Tokenizing

Turning text into smaller units, essentially spl
paragraph or entire document into smaller u
individual words, numbers, or punctuation m
for natural language processing.

In English:

- split by spaces
- more advanced algorithms

Spacy tokenizer

1. Iterate over whitespace-separated substrings.

Tokenizing with unnest_toke

```
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
## # ℹ 26,677 more rows
```

Words as a unit

Now that we have words as the observation u
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
## # ℹ 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
```

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
```

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
```

Using dplyr verbs

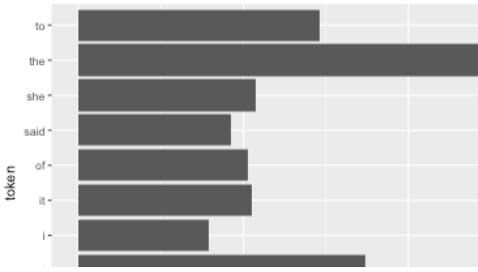
```r
library(dplyr)
alice |>
  unnest_tokens(token, text) |>
  group_by(chapter) |>
  count(token) |>
  top_n(10, n)
```

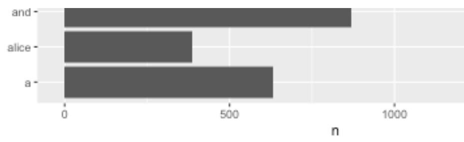
```
## # A tibble: 122 × 3
## # Groups:   chapter [12]
##    chapter token     n
##      <int> <chr> <int>
##  1       1 a        52
##  2       1 alice    27
##  3       1 and      65
##  4       1 i        30
##  5       1 it       62
```

```
## 6        1 of        43
## 7        1 she       79
## 8        1 the       92
```

Using dplyr verbs and ggplo

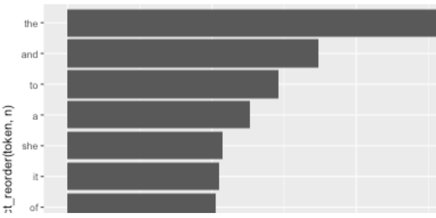
```
library(dplyr)
library(ggplot2)
alice |>
  unnest_tokens(token, text) |>
  count(token) |>
  top_n(10, n) |>
  ggplot(aes(n, token)) +
  geom_col()
```

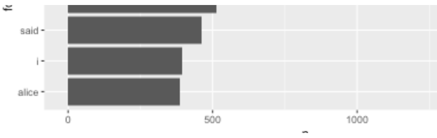




Using dplyr verbs and g

```
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. Wo
and "for" appear a lot in English text but does

Words such as these are called stop words

For more information about differences in sto
them read this chapter https://smltar.com/sto

Stop words in tidytext

tidytext comes with a data.frame of stop word

```
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
## # ℹ 1,139 more rows
```

Stopwords

```
##   [1] "a"           "about"       "above"       "across
##   [6] "again"       "against"     "all"         "almost
##  [11] "along"       "already"     "also"        "althou
##  [16] "among"       "an"          "and"         "anothe
##  [21] "anybody"     "anyone"      "anything"    "anywhe
##  [26] "area"        "areas"       "around"      "as"
##  [31] "asked"       "asking"      "asks"        "at"
##  [36] "back"        "backed"      "backing"     "backs"
##  [41] "became"      "because"     "become"      "become
##  [46] "before"      "began"       "behind"      "being"
##  [51] "best"        "better"      "between"     "big"
##  [56] "but"         "by"          "came"        "can"
##  [61] "case"        "cases"       "certain"     "certai
##  [66] "clearly"     "come"        "could"       "did"
##  [71] "different"   "differently" "do"          "does"
##  [76] "down"        "down"        "downed"      "downin
```

```
##  [81] "during"      "each"        "early"       "either
##  [86] "ended"       "ending"      "ends"        "enough
##  [91] "evenly"      "ever"        "every"       "everyb
```

Removing stopwords

We can use an `anti_join()` to remove the toke
`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
```

Anti-join with same variable

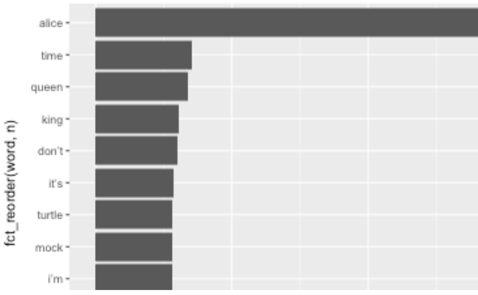
```r
alice |>
  unnest_tokens(word, text) |>
  anti_join(stop_words, by = c("word")) |>
  count(word, sort = TRUE)
```

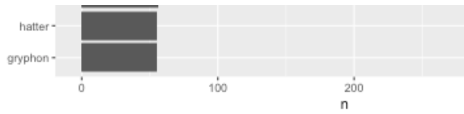
```
## # A tibble: 2,314 × 2
##    word       n
##    <chr>  <int>
##  1 alice    386
##  2 time      71
##  3 queen     68
##  4 king      61
##  5 don't     60
##  6 it's      57
##  7 i'm       56
##  8 mock      56
```

```
##  9 turtle     56
## 10 gryphon    55
## # ℹ 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
```

Which words appear togeth

ngrams are n consecutive word, we can coun
appears together.

- ngram with n = 1 are called unigrams: "w
"together"
- ngram with n = 2 are called bigrams: "wh
"appears together"
- ngram with n = 3 are called trigrams: "wh
appears together"

Which words appears toget

We can extract bigrams using `unnest_ngrams()`

```
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
```

Which words appears t

Tallying up the bi-grams still shows a lot of sto
relationships

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

```
## # A tibble: 13,424 × 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
```

Which words appears t

```
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
```

```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
## # ℹ 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
```

```
## # ℹ 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
```

```
## # ℹ 96 more rows
```

TF-IDF

TF: Term frequency gives weight to terms tha how important a word may be and how frequ document (e.g. a book chapter). IDF decrease used words and increases the weight for word in a collection of documents (e.g. all chapters

Some words that occur many times in a docu in English, these are probably words like "the" might take the approach of adding words like and removing them before analysis, but it is p words might be more important in some doc stop words is not a sophisticated approach to commonly used words.

TF-IDF

IDF: Inverse document frequency

IDF decreases the weight for commonly usec
weight for words that are not used very much

The inverse document frequency for any give

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

TF-IDF

TF-IDF: TF and IDF can be combined (the two
together), which is the frequency of a term ac

The idea of TF-IDF is to find the important wo
document by decreasing the weight for comm
increasing the weight for words that are not u
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.
## # ℹ 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
## # ℹ 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
```

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
```

Sentiment Analysis

- Sentiment Analysis is a process of extrac
different scores like positive, negative or n

- Based on sentiment analysis, you can fin
sentences in text.

- Sentiment Analysis is a type of classificat
classified into different classes like positive
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 × 2
##    word          n
##    <chr>     <int>
##  1 beautiful    13
##  2 majesty      12
##  3 glad         11
```

```
##   4 bright        8
##   5 eagerly       8
##   6 ready         8
```

Sentiment Analysis

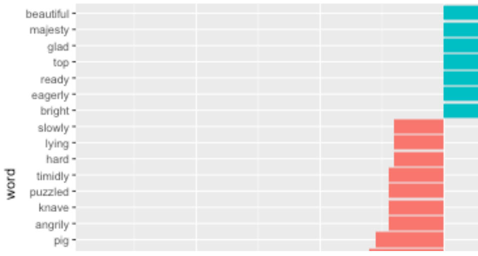
```
bing <- get_sentiments("bing")
alicesentiment<-alice |>
  unnest_tokens(word, text) |>
  anti_join(stop_words, by = c("word")) |>
  inner_join(bing) |>
  count(word, sentiment, sort = TRUE)
alicesentiment
```

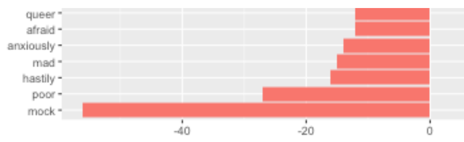
```
## # A tibble: 413 × 3
##    word      sentiment     n
##    <chr>     <chr>     <int>
##  1 mock      negative     56
##  2 poor      negative     27
##  3 hastily   negative     16
##  4 mad       negative     15
```

```
##   5 anxiously negative     14
##   6 beautiful positive     13
##   7 afraid    negative     12
```

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")
```





Topic Modeling with **`topicm`**

One method for topic modeling is Latent Diri
guided model to discover topics in a collectio
words into these topics.

For example a two-topic model of news articl
"politics". Words that would go into sports cou
"basketball", "football", etc. and those that mig
"election", "prime minister", "mayor" etc.

LDA is a mathematical method for estimatin
time: finding the mixture of words that is asso
also determining the mixture of topics that d

Topic Modeling with **topicm**

We need the `topicmodels` package as well as `t`

To apply the models, we need to create a doc
matrix where:

```
each row represents one document (such as a book or articl
each column represents one term, and
each value (typically) contains the number of appearances
```

Term-Document Matrix

```r
library(tm)
library(topicmodels)

alice_dtm <- alice |>
  unnest_tokens(token, text) |>
  anti_join(stop_words, by = c("token" = "word")) |>
  DocumentTermMatrix()

alice_dtm <- as.matrix(alice_dtm)
```

LDA

```
alice_lda <- LDA(alice_dtm, k = 4, control = list(seed =
alice_lda

alice_top_terms <-
  tidy(alice_lda, matrix = "beta") |>
  group_by(topic) |>
  slice_max(beta, n = 10) |>
  ungroup() |>
  arrange(topic, -beta)

alice_top_terms |>
  mutate(term = reorder_within(term, beta, topic)) %>%
  ggplot(aes(beta, term, fill = factor(topic))) +
  geom_col(show.legend = FALSE) +
  facet_wrap(~ topic, scales = "free") +
  scale_y_reordered()
```

Customizing stopwords

```
# new words to add
new_stops <-
  c("chapter","series_","_the","well", "way","now","illus
```

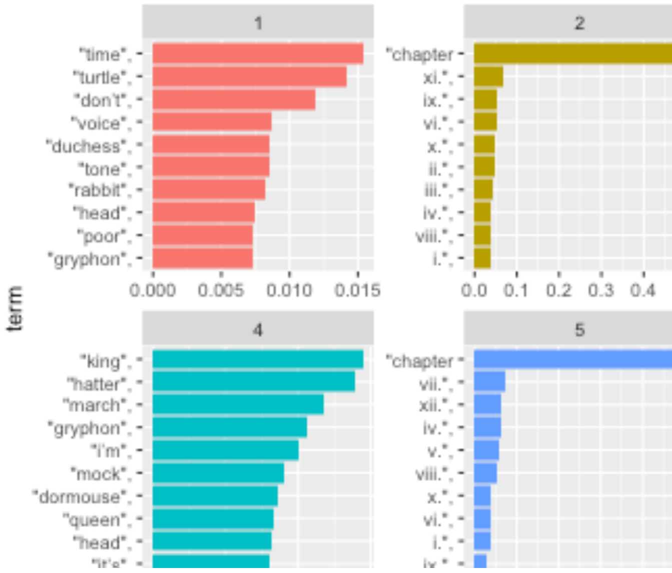
```
# need a lexicon column
custom <-
  rep("CUSTOM",length(new_stops))

# create tibble
custom_stop_words <-
  tibble(word=new_stops, lexicon=custom)

# Bind the custom stop words to stop_words
stop_words2 <-
  rbind(stop_words, custom_stop_words)
```

Term-Document Matrix revi

```
alice_dtm <- alice |>
  unnest_tokens(token, text) |>
  anti_join(stop_words2, by = c("token" = "word")) |>
  DocumentTermMatrix()
alice_dtm <- as.matrix(alice_dtm)
```

LDA revisited

```
## A LDA_VEM topic model with 6 topics.
```




0.000
0.005
0.010
0.015
0.0
0.1
0.2
0.3
0.4
beta