Sentiment Analysis

Max Turgeon

SCI 2000–Introduction to Data Science

Lecture Objectives

- Explain the pros and cons of the bag-of-words model
- Compare texts using sentiment analysis and TF-IDF

Motivation

- Last lecture, we discussed regular expressions.
- They're a way to *manipulate* text data:
    - Filter according to the presence of a pattern.
    - Replace a certain pattern.
    - Split a string into smaller components according to a pattern.
- Today we are looking at **sentiment analysis** which is a way to *analyze* text data.

Bag-of-words model i

- Let's assume we have a collection of strings.
  - E.g. a series of tweets, chapters from a book, articles on Canadian politics.
- We need a way to represent these strings so we can make comparisons.
  - E.g. is this article fake news? Is this email spam? Are these two tweet about the same topic?
- A very common representation is the **bag-of-words** model.
- Every string is represented as an (unordered) set of its words.
  - No punctuation
  - Ignoring grammar and word order

Bag-of-words model ii

- For example, let's consider the following sentence:
  - `The Queen saluted the work of front line workers across the Commonwealth.`
- It's bag-of-words representation is:
  - `"The", "Queen", "saluted", "the", "work", "of", "front", "line", "workers", "across", "the", "Commonwealth"`
- In particular, we keep repetitions.

Example i

```
library(stringr)
string <- "The Queen saluted the work of front
line workers across the Commonwealth."

bag_words <- str_split(string,
                       pattern = "\\s+")
bag_words
```

Example ii

```
## [[1]]
## [1] "The" "Queen" "saluted" "the"
## [5] "work" "of" "front" "line"
## [9] "workers" "across" "the" "Commonwealth."

# Remove the final period
# Recall: bag_words is a list
str_replace(bag_words[[1]],
            pattern = "\\.$",
            replacement = "")
```

Example iii

```
## [1] "The" "Queen" "saluted" "the" "work"
## [6] "of" "front" "line" "workers" "across"
## [11] "the" "Commonwealth"

# If we had more than one string
# we could use map from the purrr package
library(purrr)
bag_words %>%
  map(~str_replace_all(.x, "\\.$", ""))
```

Example iv

```
## [[1]]
## [1] ”The” ”Queen” ”saluted” ”the” ”work”
## [6] ”of” ”front” ”line” ”workers” ”across”
## [11] ”the” ”Commonwealth”
```

Exercise

Turn the following tweet into its bag-of-words representation:

```
We've launched the #5030Challenge to make workplaces
across the country more diverse and inclusive - because
when that happens, we all benefit.
```

Can you also remove the hashtag? (Hint: look at the function `str_subset`.)

Solution i

```
string <- "We've launched the #5030Challenge
to make workplaces across the country
more diverse and inclusive - because
when that happens, we all benefit."

bag_words <- str_split(string,
                       pattern = "\\s+")
bag_words
```

Solution ii

```
## [[1]]
## [1] "We've" "launched" "the" "#5030Challenge"
## [5] "to" "make" "workplaces" "across"
## [9] "the" "country" "more" "diverse"
## [13] "and" "inclusive" "-" "because"
## [17] "when" "that" "happens," "we"
## [21] "all" "benefit."
```

```
# Remove the final period and the comma
str_replace(bag_words[[1]],
            pattern = "(\\.|,)$",
            replacement = "")
```

Solution iii

```
## [1] "We've" "launched" "the" "#5030Challenge"
## [5] "to" "make" "workplaces" "across"
## [9] "the" "country" "more" "diverse"
## [13] "and" "inclusive" "-" "because"
## [17] "when" "that" "happens" "we"
## [21] "all" "benefit"
```

Solution iv

```
# Remove the final period and the comma
# also remove hyphen and hashtag
str_replace(bag_words[[1]],
            pattern = "(\\.|,)$",
            replacement = "") %>%
  str_subset(pattern = "-", negate = TRUE) %>%
  str_subset(pattern = "^#", negate = TRUE)
```

Solution v

```
## [1] "We've" "launched" "the" "to" "make"
## [6] "workplaces" "across" "the" "country"
"more"
## [11] "diverse" "and" "inclusive" "because"
"when"
## [16] "that" "happens" "we" "all" "benefit"
```

Question

Can you find advantages and disadvantages of the bag-of-words model?

Answer

- Advantages
  - Simplifies comparison
  - Easy to understand
- Disadvantages
  - Ignores relationship between words
  - May distort meaning (i.e. `like` and `not like`)

Tokenization and stop-words i

- More generally, the process of splitting a string into smaller components is called **tokenization**.
- Therefore, the words are sometimes called **tokens**.
- Some tokens do not provide much information about a string or text because they don't carry much meaning, or they are too common.
    - E.g. `the`, `and`, `or`, etc.
- These tokens are called **stop-words**, and they are often removed from bags-of-words.
- The dataset `stop_words` in the `tidytext` package contains a lexicon of stop-words.

Tokenization and stop-words ii

```
library(tidytext)
head(stop_words, n = 5)
```

```
## # A tibble: 5 x 2
##   word  lexicon
##   <chr> <chr>
## 1 a     SMART
## 2 a's   SMART
## 3 able  SMART
## 4 about SMART
## 5 above SMART
```

Tokenization and stop-words iii

- If we store our bag-of-words into a `data.frame`, then we can use an anti-join to remove stop-words.

Example i

```
# Previous example
bag_words <- str_replace(bag_words[[1]],
                         pattern = "(\\.|,)$",
                         replacement = "") %>%
  str_subset(pattern = "-", negate = TRUE) %>%
  str_subset(pattern = "^#", negate = TRUE)


library(tidyverse)
dataset <- data.frame(word = bag_words)
dataset %>%
  anti_join(stop_words, by = "word")
```

Example ii

```
##         word
## 1       We've
## 2   launched
## 3 workplaces
## 4    country
## 5    diverse
## 6  inclusive
## 7    benefit
```

Using tidytext i

```
# Store strings in a data.frame
# and give them an id number
dataset <- data.frame(id = c(1, 2),
  string = c(
  "The Queen saluted the work of front
line workers across the Commonwealth.",
  "We've launched the #5030Challenge
to make workplaces across the country
more diverse and inclusive - because
when that happens, we all benefit.")
)
```

Using tidytext ii

```
dataset %>%
  unnest_tokens(output = "word",
                input = "string") %>%
  glimpse
```

```
## Rows: 33
## Columns: 2
## $ id <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2,~
## $ word <chr> "the", "queen", "saluted", "the",
"work", "of", "front", "line", ~
```

Using tidytext iii

```
data_clean <- dataset %>%
  unnest_tokens(output = "word",
                input = "string") %>%
  anti_join(stop_words)
glimpse(data_clean)
```

```
## Rows: 14
## Columns: 2
## $ id <dbl> 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2,
2, 2
## $ word <chr> "queen", "saluted", "front",
"line", "workers", "commonwealth", "~
```

Sentiment analysis

- **Sentiment analysis** is a popular way of analyzing and comparing bag-of-words.
- The idea is to build a lexicon and attach a sentiment, or a sentiment value, to each word in the lexicon.
- We can then compute the most common sentiment, or average sentiment value, for a particular text.
- Fortunately, there are many lexica we can readily use!

Example i

```
# We will use the Bing lexicon
head(get_sentiments("bing"), n = 5)
```

```
## # A tibble: 5 x 2
##   word       sentiment
##   <chr>      <chr>
## 1 2-faces    negative
## 2 abnormal   negative
## 3 abolish    negative
## 4 abominable negative
## 5 abominably negative
```

Example ii

```
# Why are we using an inner join?
data_clean %>%
  inner_join(get_sentiments("bing"),
             by = "word") %>%
  count(id, sentiment)
```

```
##   id sentiment n
## 1  2  positive 1
```

```
# There was only one word in both strings
# that appeared in the Bing lexicon...
```

Example i

```
# We will analyse a larger corpus
# Anne of Green Gables
# We can download it from the project Gutenberg
# gutenberg_id = 45 is the book we want
library(gutenbergr)
full_text <- gutenberg_download(45)
glimpse(full_text)
```

Example ii

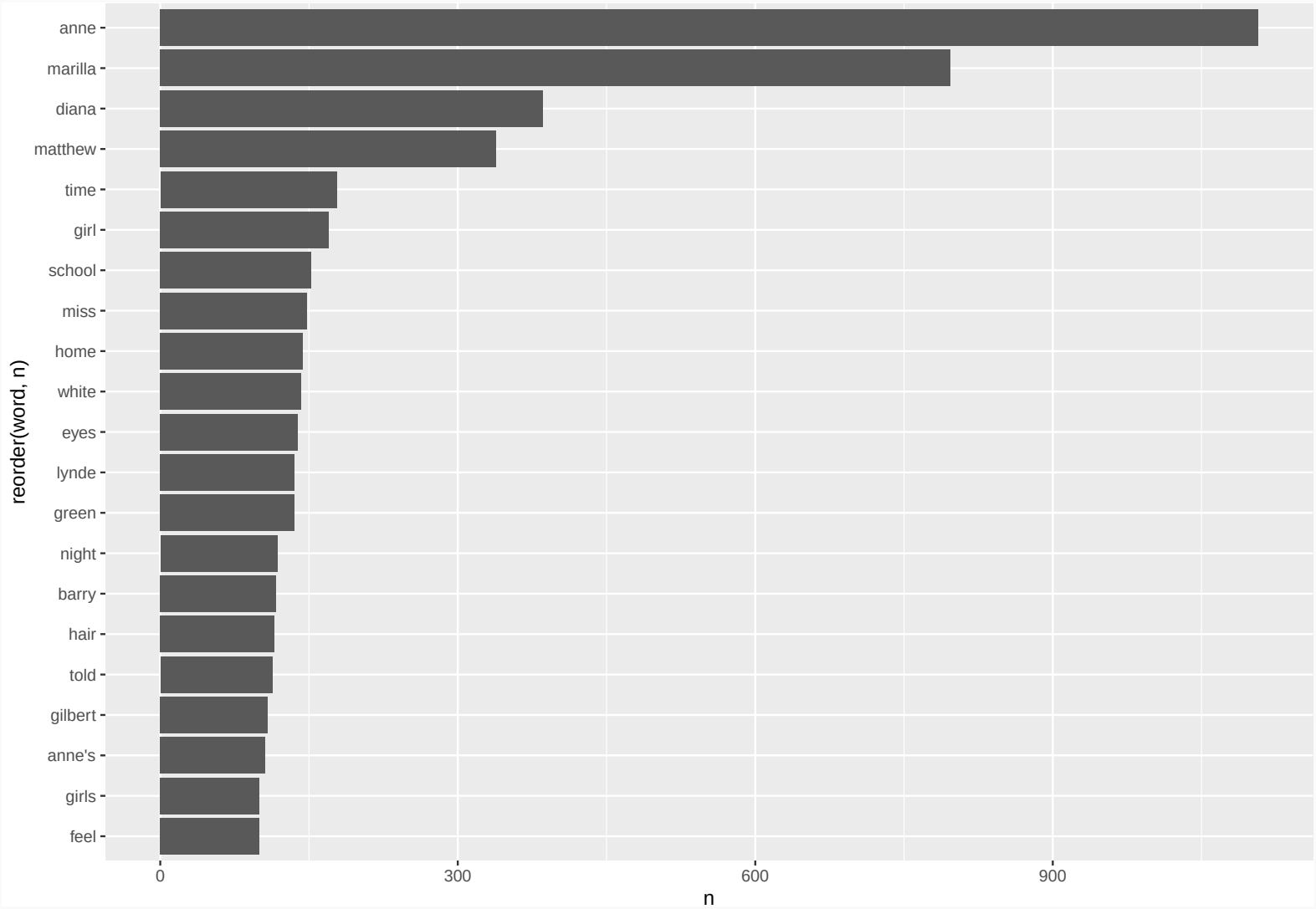
```
## Rows: 10,779
## Columns: 2
## $ gutenberg_id <int> 45, 45, 45, 45, 45, 45,
45, 45, 45, 45, 45, 45, 45, 45, 4~
## $ text <chr> "ANNE OF GREEN GABLES", "", "By
Lucy Maud Montgomery", ""~
```

```
# ID each line, tokenize, and clean
data_clean <- full_text %>%
  mutate(line = row_number()) %>%
  unnest_tokens(word, text) %>%
  anti_join(stop_words, by = "word")
```

Example iii

```
# Visualize most common words
data_clean %>%
    count(word, sort = TRUE) %>%
    top_n(n = 20) %>%
    ggplot(aes(n, reorder(word, n))) +
    geom_col()
```

Example iv


anne
marilla
diana
matthew
time
girl
school
miss
home
white
eyes
lynde
green
night
barry
hair
told
gilbert
anne's
girls
feel
reorder(word, n)
0
300
600
900
n

Example v

```
# Sentiment analysis----
data_sentiment <- data_clean %>%
  inner_join(get_sentiments("bing"),
             by = "word") %>%
  count(sentiment, word, sort = TRUE)
# We lost a lot of words...
c(nrow(data_clean), nrow(data_sentiment))
```

```
## [1] 34186  1427
```

Example vi

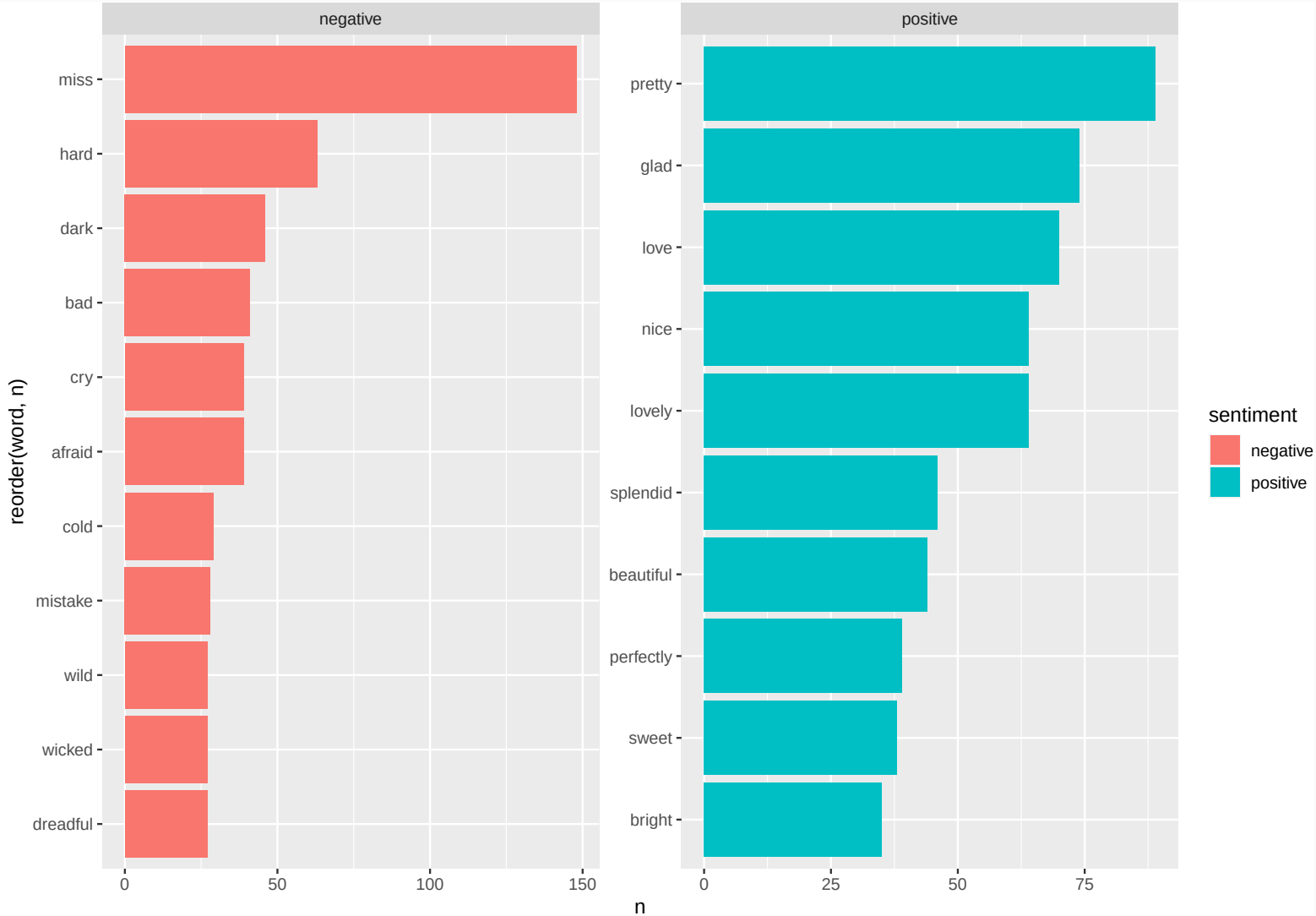
```
# Let's look at a few rows
head(data_sentiment)
```

```
## # A tibble: 6 x 3
##   sentiment word        n
##   <chr>     <chr>   <int>
## 1 negative  miss      148
## 2 positive  pretty     89
## 3 positive  glad       74
## 4 positive  love       70
## 5 positive  lovely     64
## 6 positive  nice       64
```

Example vii

```
data_sentiment %>%
  group_by(sentiment) %>%
  top_n(n = 10) %>%
  ggplot(aes(n, reorder(word, n),
             fill = sentiment)) +
  geom_col() +
  facet_wrap(~ sentiment, scales = "free")
```

Example viii



Exercise

Repeat the analysis for *The Wonderful Wizard of Oz* (`gutenberg_id = 55`). What are the most common positive and negative words?

Solution i

```
# Download full text,
# ID each line, tokenize, and clean
data_clean <- gutenberg_download(55) %>%
  mutate(line = row_number()) %>%
  unnest_tokens(word, text) %>%
  anti_join(stop_words, by = "word")
```

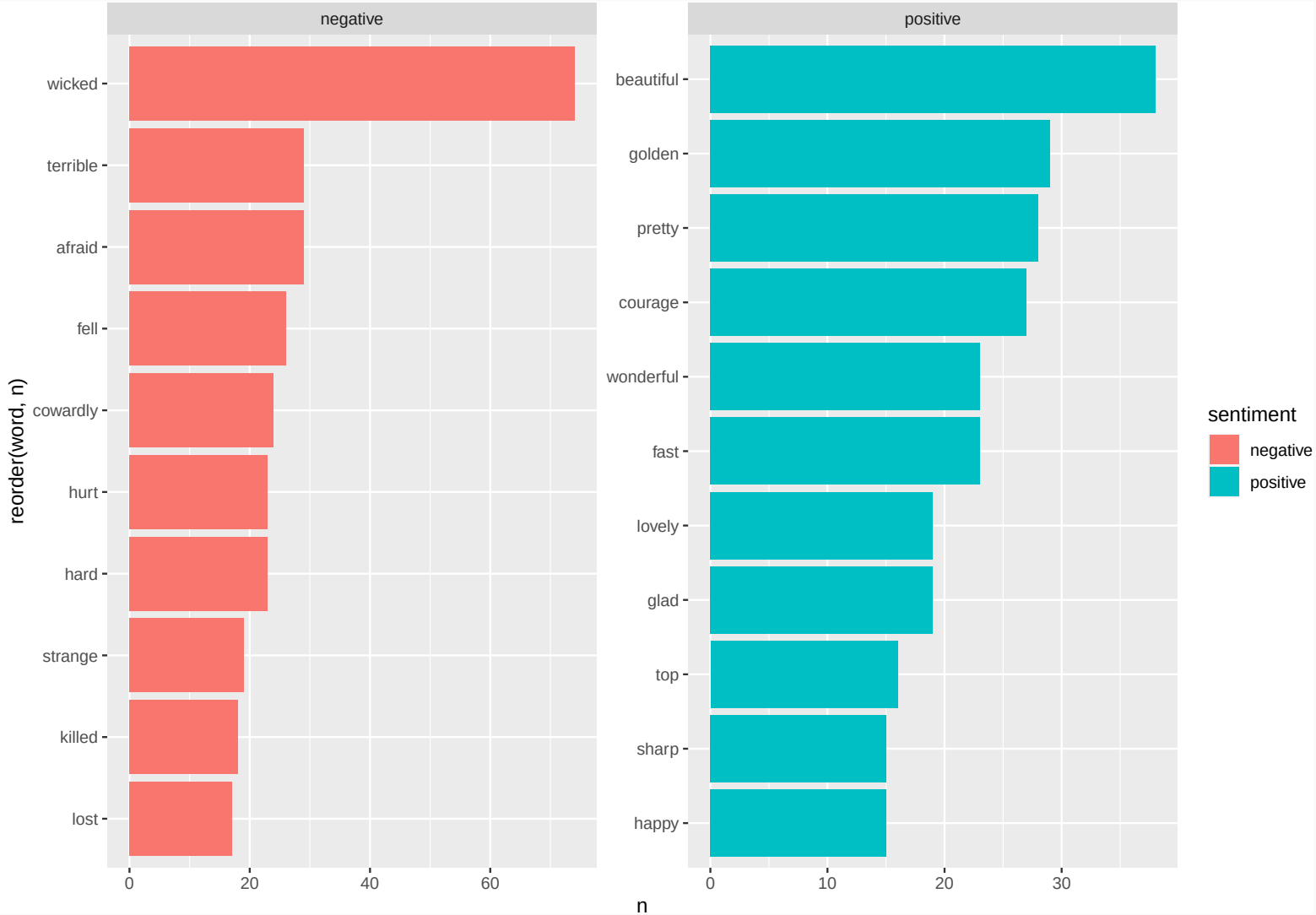
Solution ii

```
# Sentiment analysis----
data_sentiment <- data_clean %>%
  inner_join(get_sentiments("bing"),
             by = "word") %>%
  count(sentiment, word, sort = TRUE)
```

Solution iii

```
data_sentiment %>%
  group_by(sentiment) %>%
  top_n(n = 10) %>%
  ggplot(aes(n, reorder(word, n),
             fill = sentiment)) +
  geom_col() +
  facet_wrap(~ sentiment, scales = "free")
```

Solution iv


negative
positive
wicked
terrible
afraid
fell
cowardly
hurt
hard
strange
killed
lost
beautiful
golden
pretty
courage
wonderful
fast
lovely
glad
top
sharp
happy
reorder(word, n)
n
0
20
40
60
0
10
20
30
sentiment
negative
positive

TF-IDF

- Sentiment analysis is not the only way to turn words into numbers/values.
- TF-IDF is another approach:
  - **Term Frequency**: How many times a word appears in a document
  - **Inverse Document Frequency**: Negative log of fraction of documents containing a certain word.
- Taking the product of these two quantities, TF-IDF allows us to measure the important of a particular word within a collection of documents.
- In particular, we don't need to remove stop-words; they'll have `IDF = 0`

Example i

```
library(tidytext)
dataset <- data.frame(id = c(1, 2),
  string = c(
  "The Queen saluted the work of front
line workers across the Commonwealth.",
  "We've launched the #5030Challenge
to make workplaces across the country
more diverse and inclusive - because
when that happens, we all benefit.")
)
```

Example ii

```
data_tfidf <- dataset %>%
  unnest_tokens(output = "word",
                input = "string") %>%
  count(id, word) %>%
  bind_tf_idf(term = word,
              document = id,
              n = n)

head(data_tfidf, n = 5)
```

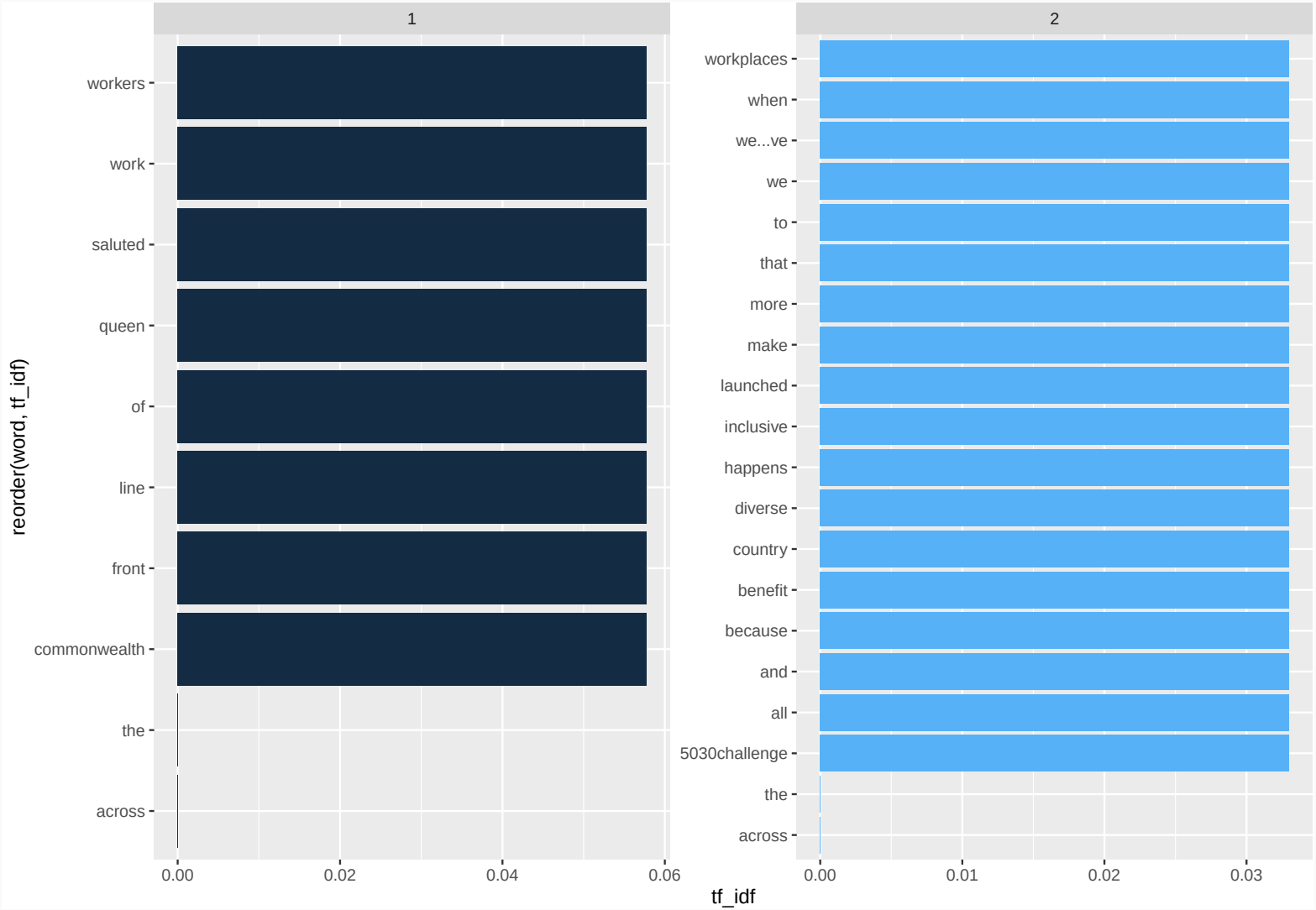
Example iii

```
##   id          word n    tf  idf tf_idf
## 1  1        across 1 0.083 0.00  0.000
## 2  1 commonwealth 1 0.083 0.69  0.058
## 3  1         front 1 0.083 0.69  0.058
## 4  1          line 1 0.083 0.69  0.058
## 5  1            of 1 0.083 0.69  0.058
```

Example iv

```
data_tfidf %>%
    ggplot(aes(tf_idf,
               reorder(word, tf_idf),
               fill = id)) +
    geom_col(show.legend = FALSE) +
    facet_wrap(~id, scales = "free")
```

Example v


1
2
workers
work
saluted
queen
of
line
front
commonwealth
the
across
workplaces
when
we...ve
we
to
that
more
make
launched
inclusive
happens
diverse
country
benefit
because
and
all
5030challenge
the
across
reorder(word, tf_idf)
0.00
0.02
0.04
0.06
0.00
0.01
0.02
0.03
tf_idf

Exercise

Repeat the analysis with the first three novels from the Anne of Green Gables series. What are the top words for each novel, according to TF-IDF? You can start with the code below.

```
library(tidytext)
library(gutenbergr)
anne_novels <- gutenberg_download(c(45, 47, 51),
                                  meta_fields ="title")
```

Solution i

```
anne_novels <- gutenberg_download(c(45, 47, 51),
                                  meta_fields = "title")

anne_novels
```

```
## # A tibble: 29,389 x 3
## gutenberg_id text title
## <int> <chr> <chr>
## 1 45 "ANNE OF GREEN GABLES" Anne of Green Ga~
## 2 45 "" Anne of Green Ga~
## 3 45 "By Lucy Maud Montgomery" Anne of Green
Ga~
```

Solution ii

```
## 4 45 "" Anne of Green Ga~
## 5 45 "" Anne of Green Ga~
## 6 45 "" Anne of Green Ga~
## 7 45 "Table of Contents" Anne of Green Ga~
## 8 45 "" Anne of Green Ga~
## 9 45 " CHAPTER I Mrs. Rachel Lynde Is~ Anne of
Green Ga~
## 10 45 " CHAPTER II Matthew Cuthbert Is ~ Anne
of Green Ga~
## # ... with 29,379 more rows
```

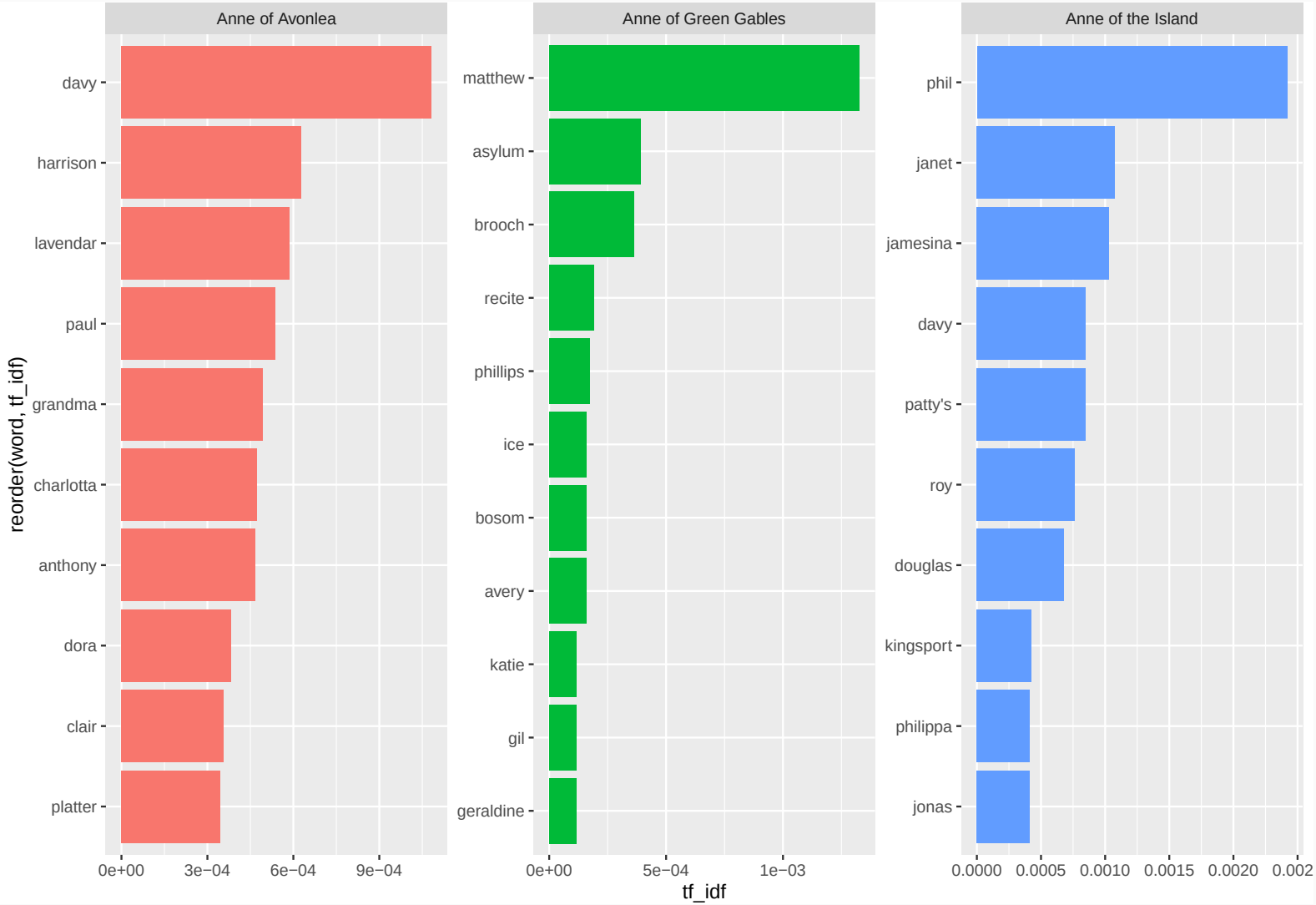
Solution iii

```
data_tfidf <- anne_novels %>%
  unnest_tokens(word, text) %>%
  count(title, word) %>%
  bind_tf_idf(word, title, n)
```

Solution iv

```
# Visualize top 10
data_tfidf %>%
    group_by(title) %>%
    top_n(n = 10, wt = tf_idf) %>%
    ggplot(aes(tf_idf,
               reorder(word, tf_idf),
               fill = title)) +
    geom_col(show.legend = FALSE) +
    facet_wrap(~title, scales = "free")
```

Solution v


Anne of Avonlea
Anne of Green Gables
Anne of the Island
reorder(word, tf_idf)
davy
harrison
lavendar
paul
grandma
charlotta
anthony
dora
clair
platter
matthew
asylum
brooch
recite
phillips
ice
bosom
avery
katie
gil
geraldine
phil
janet
jamesina
davy
patty's
roy
douglas
kingsport
philippa
jonas
0e+00
3e−04
6e−04
9e−04
0e+00
5e−04
1e−03
0.0000
0.0005
0.0010
0.0015
0.0020
0.002
tf_idf

Summary i

- We defined at the bag-of-words model for text data.
- We looked at two different analytic approaches:
  - Sentiment analysis
  - TF-IDF
- TF-IDF can be used to build document-term matrices.
  - These matrices are inputs for **topic modeling** and **semantic analysis**.
- Another common data manipulation is **lemmatization**: turn inflected words into a common representative.
  - E.g. `liked`, `likes`, and `likeable` would be represented by `like`.

Summary ii

- Instead of bag-of-words, we can use **N-grams**: tokens are pairs/triples/tuples of consecutive words.
- Finally, a new branch of text analysis uses **neural networks** to construct predictive models for text.
    - E.g. Predictive text on your phone
- As you can see, there is a lot to explore, and I hope this lecture was enough to capture your interest!