Relationship between words

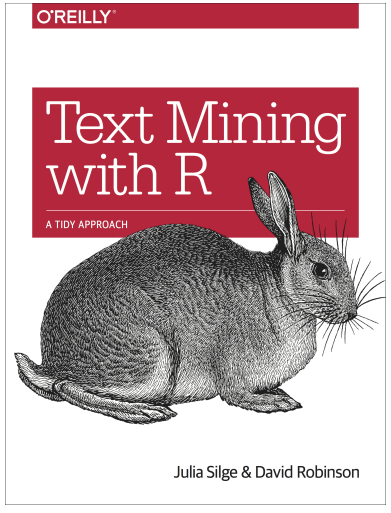
Rei Sanchez-Arias, Ph.D.

Tokenizing by n-grams, analyzing bigrams, and using n-grams in sentiment analysis

Relations between words

Co-occurrence of words

"Text Mining with R: A Tidy Approach" by Julia Silge and David Robinson




```
library(tidytext)
library(tidyverse)
```

Many interesting text analyses are based on the **relationships between words**, whether examining which words *tend* to **follow** others immediately, or that *tend* to **co-occur** within the same documents.

> Examples and materials in this set of slides are adapted from the book by Silge and Robinson

Tokenizing by n-gram

We will use the `token = "ngrams"` argument in the `unnest_tokens()`, which tokenizes by pairs of adjacent words rather than by individual ones. This allows us to tokenize into consecutive sequences of words, called **n-grams**.

By seeing how often word `X` is followed by word `Y`, we can then build a model of the relationships between them. The parameter `n` is the number of words we wish to capture in each n-gram, and we will start by setting `n` to 2, to indicate we are examining ***pairs of two consecutive words***, often called **bigrams**:

```
rmp <- read_csv("https://raw.githubusercontent.com/reisanar/datasets/master/rmp_wit_comments.c
```

Bigrams

```
rmp_bigrams <- rmp %>%
  unnest_tokens(bigram, comments,
                token = "ngrams", n = 2)
```

	course	bigram
1	MATH1900	he is
2	MATH1900	is very
3	MATH1900	very enthusiastic
4	MATH1900	enthusiastic to

Showing 1 to 4 of 829 entries

Previous 1 2 3 4 5 ... 208 Next

Counting and filtering n-grams

Our usual tidy tools apply equally well to n-gram analysis. We can examine the most common bigrams using `dplyr`'s `count()`:

```
rmp_bigrams %>%
  count(bigram, sort = TRUE)
```

	bigram	n
1	he is	14
2	great professor	8
3	is a	7

Previous 1 2 3 4 5 ... 239 Next

As one might expect, a lot of the most common bigrams are pairs of common (uninteresting) words, such as `to a`: what we call stop-words

Counting and filtering n-grams (cont.)

We can use `tidyr`'s `separate()` function, which splits a column into multiple based on a delimiter. This lets us separate it into two columns, say `word1` and `word2`, at which point we can remove cases where either is a stop-word.

```
rmp_bigrams %>%
  separate(bigram, into = c("word1", "word2"), sep = " ")
```

	course	word1	word2
1	MATH1900	he	is
2	MATH1900	is	very
3	MATH1900	very	enthusiastic

Previous 1 2 3 4 5 ... 277 Next

Remove stop-words

```
bigrams_filtered <- rmp_bigrams %>%
    separate(bigram, into = c("word1", "word2"), sep = " ") %>%
    filter(!word1 %in% stop_words$word) %>%
    filter(!word2 %in% stop_words$word)
# new bigram counts:
bigram_counts <- bigrams_filtered %>%
  count(word1, word2, sort = TRUE)
```

	word1	word2	n
1	pre	calc	3
2	linear	algebra	2
3	makes	class	2

Previous 1 2 3 4 5 ... 22 Next

Remove stop-words (cont.)

In other analyses, we may want to work with the recombined words. `tidyr`'s `unite()` function is the inverse of `separate()`, and allows us to recombine the columns into one.

```
bigrams_united <- bigrams_filtered %>%
  unite(bigram, word1, word2, sep = " ")
```

	course	bigram
1	MATH1900	email communication
2	MATH1900	office hour
3	MATH1900	awesome teacher
4	MATH2025	class entertained

Previous 1 2 3 4 5 ... 18 Next

Trigrams

In other analyses you may be interested in the most common **trigrams**, which are consecutive sequences of 3 words. We can find this by setting `n = 3`:

```
rmp %>%
  unnest_tokens(trigram, comments, token = "ngrams", n = 3) %>%
  separate(trigram, c("word1", "word2", "word3"), sep = " ") %>%
  filter(!word1 %in% stop_words$word,
         !word2 %in% stop_words$word, !word3 %in% stop_words$word) %>%
  count(word1, word2, word3, sort = TRUE)
```

	word1	word2	word3	n
1	communicator	powerpoint	lectures	1
2	easy	short	online	1
3	funny	explains	concepts	1

Previous 1 2 3 4 Next

Analyzing bigrams

tf-idf

A bigram can also be treated as a *term* in a document in the same way that we treated individual words. For example, we can look at the `tf-idf` of bigrams.

```
bigram_tf_idf <- bigrams_united %>%
  count(course, bigram) %>%
  bind_tf_idf(bigram, course, n) %>%
  arrange(desc(tf_idf))
```

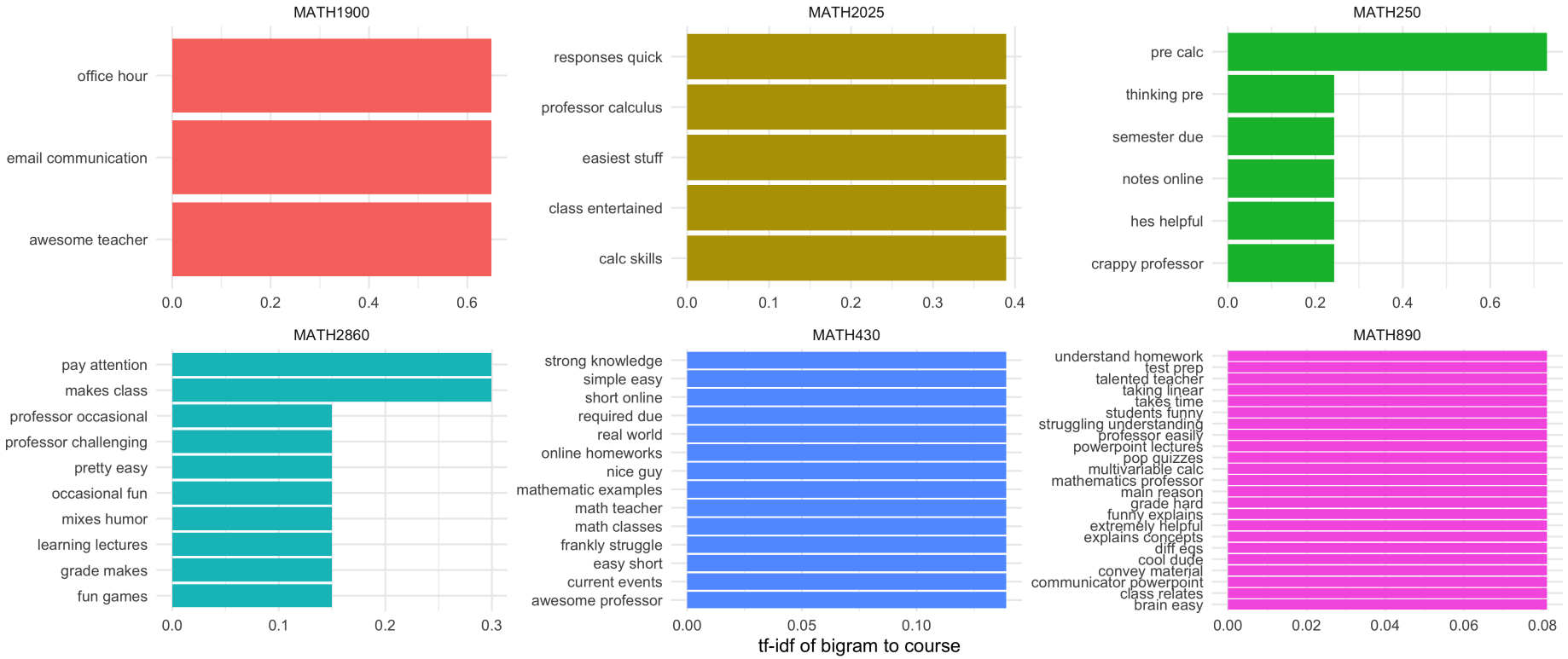
	course	bigram	n	tf_idf
1	MATH310	favorite professors	1	0.972955074527657
2	MATH310	funny guy	1	0.972955074527657
3	MATH250	pre calc	3	0.729716305895742

Showing 1 to 3 of 65 entries

Previous 1 2 3 4 5 ... 22 Next

Bigrams with the highest tf-idf

Visualization R Code Comments



Bigrams in Sentiment Analysis

Motivation

One of the problems with sentiment analysis using uni-grams, is illustrated below:

For example, the words "happy" and "like" will be counted as positive, even in a sentence like *"I'm not happy and I don't like it!"*

```
bigrams_separated <- rmp_bigrams %>%
  separate(bigram, into = c("word1", "word2"), sep = " ")
```

Now that we have the data organized into bigrams, it is easy to tell how often words are preceded by a word like "not"

Motivation (cont.)

```
bigrams_separated %>%
  filter(word1 == "not") %>%
  count(word1, word2, sort = TRUE)
```

	word1	word2	n
1	not	a	1
2	not	difficult	1
3	not	often	1
4	not	one	1
5	not	only	1
6	not	really	1

Previous 1 Next

Sentiment Analysis

By performing sentiment analysis on the bigram data, we can examine how often sentiment-associated words are preceded by "not" or other negating words. We could use this to ignore or even *reverse* their contribution to the sentiment score.

Let us use the AFINN lexicon for sentiment analysis (recall that this lexicon gives a numeric sentiment score for each word, with positive or negative numbers indicating the direction of the sentiment).

```
AFINN <- get_sentiments("afinn")
# print a sample of words and scores
sample_n(AFINN, 5)
```

```
## # A tibble: 5 x 2
##   word        value
##   <chr>       <dbl>
## 1 pardons         2
## 2 deafening      -1
## 3 thank           2
## 4 afraid         -2
## 5 propaganda     -2
```

Using AFINN lexicon

We can then examine the most frequent words that were preceded by "not" and were associated with a sentiment.

```
bb_lyrics <- read_csv("https://raw.githubusercontent.com/reisanar/datasets/mas

# bigrams count
not_words <- bb_lyrics %>%
  unnest_tokens(bigram, Lyrics, token = "ngrams", n = 2) %>%
  separate(bigram, into = c("word1", "word2"), sep = " ") %>%
  filter(word1 == "not") %>%
  inner_join(AFINN, by = c(word2 = "word")) %>%
  count(word2, value, sort = TRUE)
```

Using the AFINN lexicon (cont.)

For example, the most common sentiment-associated word to follow *"not"* was *"leave"*, which would normally have a (negative) score of -1.

	word2	value	n
1	leave	-1	4
2	promised	1	2
3	scared	-2	2
4	alive	1	1

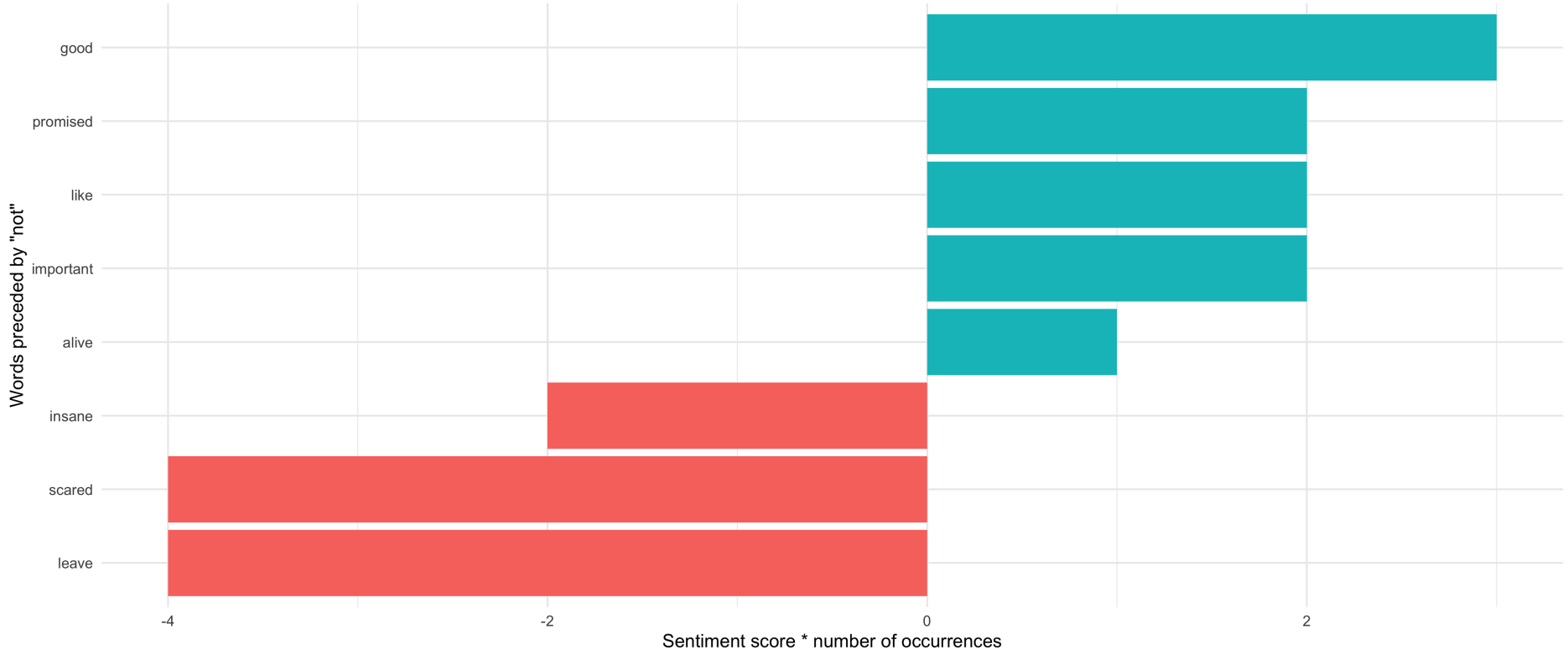
Showing 1 to 4 of 8 entries

Previous 1 2 Next

Notice that *"like"* also appears, which would have a (positive) score of 2.

Analysis with bigrams example

Visualization R Code Comments



Additional negation words

Other words

"Not" is not the only term that provides some context for the following word. We could pick four common words (or more) that negate the subsequent term, and use the same joining and counting approach to examine all of them at once.

```
negation_words <- c("not", "no", "never", "without")
```

Example: Top 100 songs in 2015

```
negated_words <- bb_lyrics %>%
  unnest_tokens(bigram, Lyrics, token = "ngrams", n = 2) %>%
  separate(bigram, into = c("word1", "word2"), sep = " ") %>%
  filter(word1 %in% negation_words) %>%
  inner_join(AFINN, by = c(word2 = "word")) %>%
  count(word1, word2, value, sort = TRUE)
```

Negate words

	word1	word2	value	n
1	no	no	-1	5
2	not	leave	-1	4
3	no	disrespect	-2	3
4	never	broken	-1	2
5	never	drop	-1	2
6	no	fool	-2	2
7	no	regrets	-2	2
8	not	promised	1	2

Previous 1 2 3 4 Next

These are just a few examples of how finding consecutive words can give context to text mining methods.

Visualizing results

Visualization R Code