

Text Mining

Text Data

Text data is one of the most commonly used types of unstructured data.

Text data is typically generated by users themselves.

Online reviews, movie critics, Tweets, SMS, WhatsApp and WeChat messages, Facebook messages...

Packages that will be used today

- dplyr
- tidytext
- tidyr
- textdata
- ggplot2
- janeaustenr
- stringr
- syuzhet
- wordcloud
- RColorBrewer

Word Frequency

Let's create some text in R.

You don't need to understand this; it is just used for demonstration.

```
1 text <- c("Hong Kong Island, known for its dazzling skyline, vibrant culture, and rich history,  
is a captivating destination nestled on the southeastern coast of China. With its status as a  
Special Administrative Region, Hong Kong Island stands as a remarkable blend of Eastern and  
Western influences, creating a unique and dynamic urban landscape. As one of the two main  
regions that make up the territory of Hong Kong, alongside the Kowloon Peninsula, Hong Kong  
Island is renowned for its cosmopolitan atmosphere, bustling streets, and iconic landmarks.",  
2 "The island's story is deeply intertwined with the history of Hong Kong itself.  
Originally a sparsely populated area, it gradually transformed into a thriving trading port  
during the 19th century, attracting merchants from around the world. Today, Hong Kong Island  
embodies the city's economic prowess, with its central business district serving as a global  
financial hub and a symbol of its economic significance.",  
3 "The island's skyline is dominated by towering skyscrapers that showcase architectural  
marvels, blending modernity with traditional Chinese motifs. Among the most prominent landmarks  
is the famous Hong Kong Convention and Exhibition Centre, an architectural gem situated on the  
waterfront. The towering Bank of China Tower and the striking International Finance Centre  
further contribute to the island's impressive skyline.",  
4 "Beyond its urban splendor, Hong Kong Island offers a diverse range of experiences.  
The district of Central is a paradise for shopaholics, boasting luxury boutiques, international  
brands, and bustling street markets. The vibrant neighborhoods of Sheung Wan and Sai Ying Pun  
offer a glimpse into Hong Kong's colonial past with their quaint streets, antique shops, and  
historic temples.")
```

Next, we transform the vector into a data frame.

```
1 library(dplyr)
2 library(tidytext)
3 text_df <- data_frame(line = 1:4, text = text)
4 text_df
```

Next, we “tokenize” the text, i.e., breaking down a sequence of text into smaller units called tokens.

```
1 mytext <- text_df %>% unnest_tokens(word, text)
```

Here, `%>%` is an operation in the `dplyr` package, which applies the tokenization function to the `text_df` data frame.

Then, we count the most frequent words in our input text.

```
1 mytext %>% count(word, sort = TRUE)
```

```
# A tibble: 160 × 2
  word      n
  <chr> <int>
1 the      18
2 a         11
3 and       10
4 of        10
5 hong       9
6 kong       8
7 is         6
8 its        6
9 island     5
10 with      5
```

Any issues with the analysis so far?

Then, we count the most frequent words in our input text.

```
1 mytext %>% count(word, sort = TRUE)
```

```
# A tibble: 160 × 2
  word      n
  <chr> <int>
1 the      18
2 a         11
3 and       10
4 of        10
5 hong       9
6 kong       8
7 is         6
8 its        6
9 island     5
10 with      5
```

Any issues with the analysis so far?

The issue is many of the frequent words are meaningless ones such as “the”, “a”. These words appear in any text and do not carry specific meaning. They are called “**stop words**” in English.

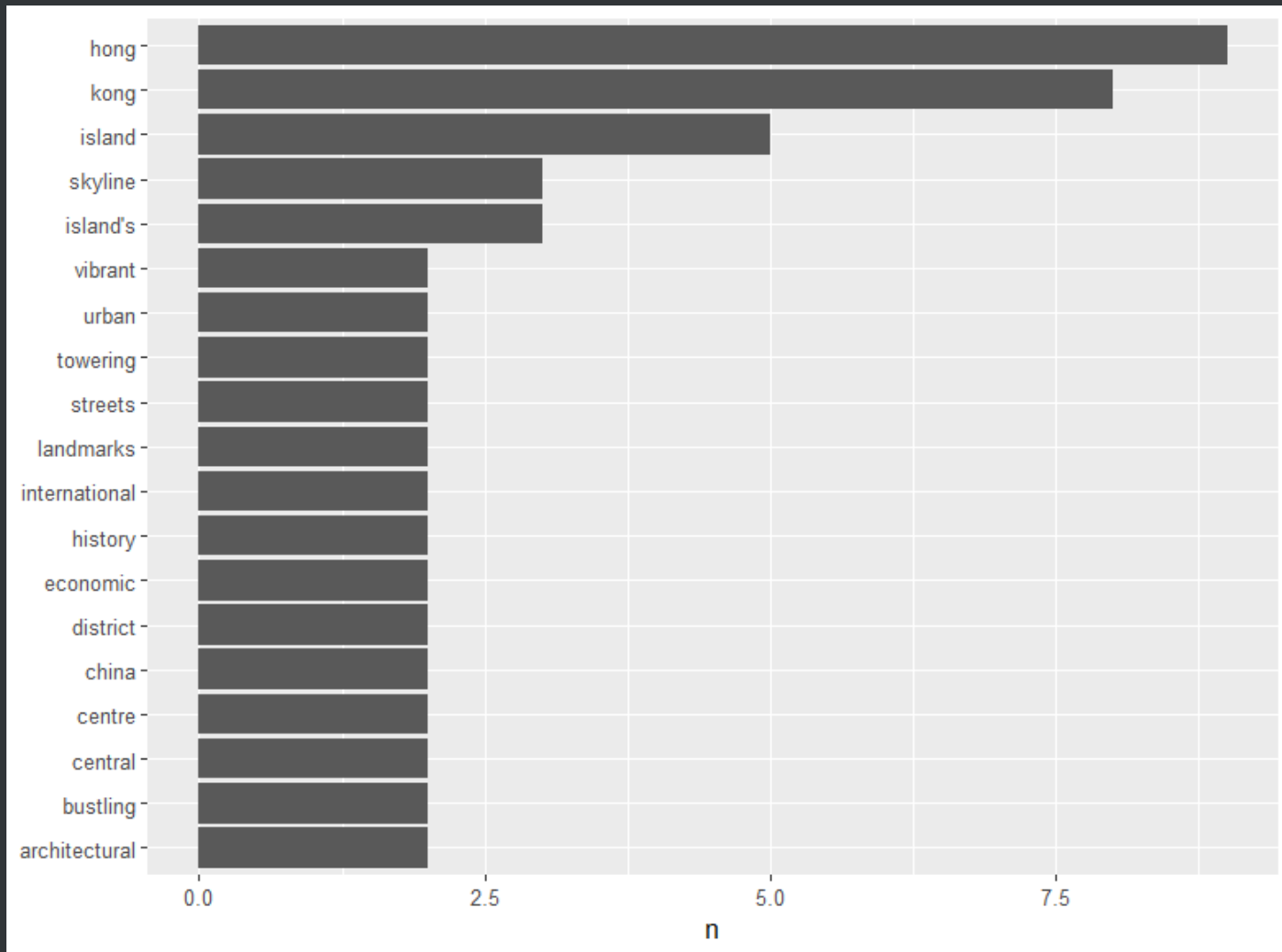
We want to remove the stop words from the text.

```
1 data(stop_words)
2 mytext <- mytext %>% anti_join(stop_words)
3 mytext %>% count(word, sort = TRUE)
```

Try the code yourself and see if it makes sense to you!

Let's visualize the most frequent words!

```
1 library(ggplot2)
2 mytext %>%
3   count(word, sort = TRUE) %>%      #sorting the words based on frequency
4   filter(n > 1) %>%                 #display words that appear more than once
5   mutate(word = reorder(word, n)) %>%
6   ggplot(aes(word, n)) +
7   geom_col() +
8   xlab(NULL) +
9   coord_flip()
```

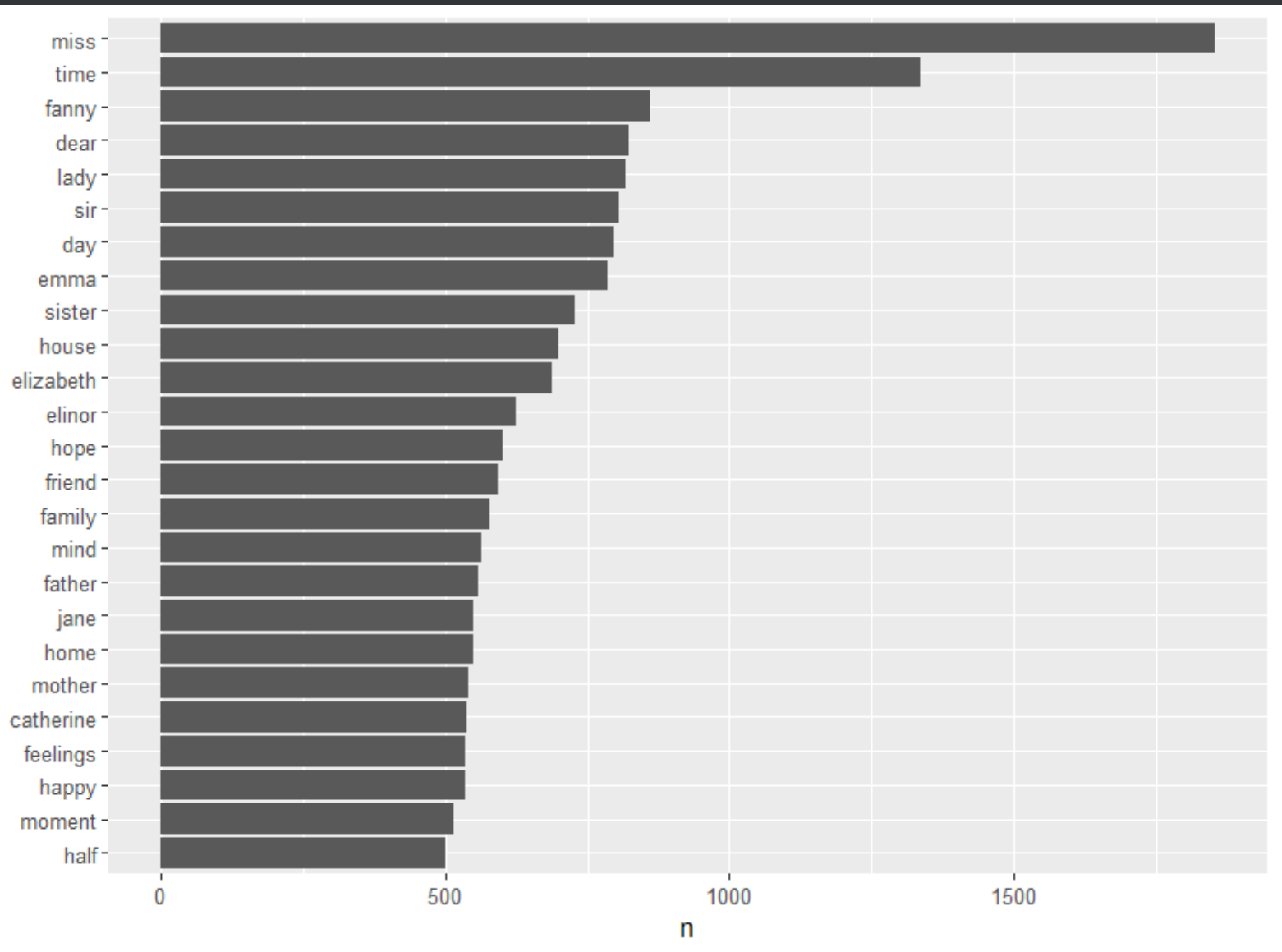


The complete code is here.

```
1 library(dplyr)
2 library(tidytext)
3 library(ggplot2)
4 data(stop_words)
5 text_df <- data_frame(line = 1:4, text = text)
6 mytext <- text_df %>% unnest_tokens(word, text)
7 mytext <- mytext %>% anti_join(stop_words)
8 mytext %>%
9   count(word, sort = TRUE) %>%      #sorting the words based on frequency
10  filter(n > 1) %>%                  #display words that appear more than once
11  mutate(word = reorder(word, n)) %>%
12  ggplot(aes(word, n)) +
13  geom_col() +
14  xlab(NULL) +
15  coord_flip()
```

In another example, we analyze the distribution of most frequent words in works of Jane Austen

```
1 library(dplyr)
2 library(tidytext)
3 library(ggplot2)
4 library(janeaustenr)
5 data(stop_words)
6 original_books <- austen_books()
7 mytext <- original_books %>%
8   unnest_tokens(word, text)
9 mytext <- mytext %>% anti_join(stop_words)
10 mytext %>%
11   count(word, sort = TRUE) %>%
12   filter(n > 500) %>%
13   mutate(word = reorder(word, n)) %>%
14   ggplot(aes(word, n)) +
15   geom_col() +
16   xlab(NULL) +
17   coord_flip()
```



We can further generate a word cloud of the frequent words:

```
1 library(wordcloud)
2 mytext %>%
3   anti_join(stop_words) %>%
4   count(word) %>%
5   with(wordcloud(word, n, max.words = 100))
```

sister
crawford pleasure miss
captain obliged looked love
edmund answer morning
manner short john time subject
minutes marianne moment feelings
anne darcy replied affection walk elinor brother
woman sort leave mother party
visits speak heard till perfectly comfort immediately
day half weston cried happy eyes friends doubt life
passed weston acquaintance sir father
brought opinion elton heart word jane deal
return spirits coming character poor rest family dear
attention mind harriet hour
left fanny emma found
idea woodhouse letter hear people
happiness home people
house friend knightley
suppose bennet
elizabeth
catherine
lady
thomas
world
told

Adding colors to the word cloud:

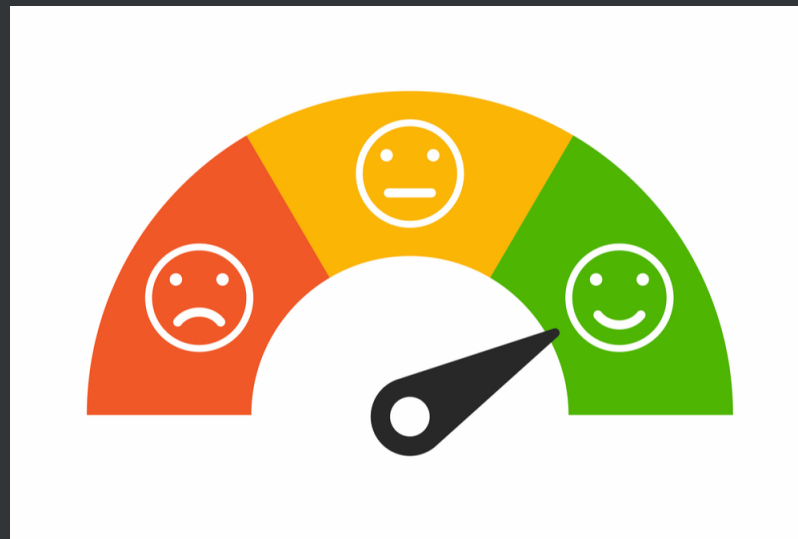
```
1 library(wordcloud)
2 library(RColorBrewer)
3
4 mytext %>%
5   anti_join(stop_words) %>%
6   count(word) %>%
7   with(wordcloud(word, n, max.words = 100,
8                 colors = brewer.pal(3, "Blues")))
```


We choose three blue colors.



Sentiment Analysis

Sentiment Analysis is arguably the most important type of text analysis. Basically, we want to classify text based on the *valence*, which can be either positive or negative (sometimes it can also be neutral).



 Finextra

Quant trader turns to reddit for sentiment forecaster

New York-based quantitative hedge fund Cindicator Capital is advertising for an active member of the wallstreetbets subreddit community to ...

3 weeks ago



Business Wire

Join the Swarm of Retail Investors Driving Sentiment. New ...

An investment in VanEck Vectors® Social Sentiment ETF (BUZZ) may be ... participant concentration, new fund, absence of prior active market, ...

5 days ago

基于情感分析的交易策略：加密对冲基金如何利用AI实现绝对收益能力

Question: In your own opinion, how should we perform sentiment analysis?

Simple Methods

The syuzhet package is a simple R package which allows you to perform sentiment analysis.

```
1 library("syuzhet")
2 text = "HKU is a fantastic school, I love it."
3 syuzhet_vector <- get_sentiment(text, method="syuzhet")
4 head(syuzhet_vector)
```

A positive (negative) output implies positive (negative) sentiment.

Simple Methods

You can perform sentiment analysis with other lexicons.

```
1 text = "HKU is a nice school and I like it."  
2 bing_vector <- get_sentiment(text, method="bing")  
3 head(bing_vector)  
4  
5 afinn_vector <- get_sentiment(text, method="afinn")  
6 head(afinn_vector)
```


Simple Methods

We can go beyond sentiment analysis to find out other emotions as well.

```
1 text = "HKU is a terrible school."  
2 print(get_nrc_sentiment(text))
```

The Sentiments Dataset

The tidytext package contains several sentiment lexicons in the sentiments dataset. To visualize the datasets, try the following code:

```
1 library(tidytext)
2 library(textdata)
3 sentiments
```

The Sentiments Dataset

The tidytext package contains three general-purpose lexicons, namely

- AFINN from Finn Årup Nielsen
- Bing from Bing Liu and collaborators
- NRC from Saif Mohammad and Peter Turney

View the lexicons:

```
1 library(tidytext)
2 library(textdata)
3 get_sentiments("afinn")
4 afinn <- get_sentiments("afinn")
5 print(afinn, n = 100)
```

Enter “1” to download the lexicons.

```
> get_sentiments("afinn")
Do you want to download:
  Name: AFINN-111
  URL: http://www2.imm.dtu.dk/pubdb/views/publication_details.php?id=6010
  License: Open Database License (ODbL) v1.0
  Size: 78 KB (cleaned 59 KB)
  Download mechanism: https

1: Yes
2: No

selection: 1
```

	word <chr>	value <dbl>
1	abandon	-2
2	abandoned	-2
3	abandons	-2
4	abducted	-2
5	abduction	-2
6	abductions	-2
7	abhor	-3
8	abhorred	-3
9	abhorrent	-3
10	abhors	-3
11	abilities	2
12	ability	2

```
1 bing <- get_sentiments("bing")
2 print(bing, n = 100)
```

1	2-faces	negative
2	abnormal	negative
3	abolish	negative
4	abominable	negative
5	abominably	negative
6	abominate	negative
7	abomination	negative
8	abort	negative
9	aborted	negative
10	aborts	negative
11	abound	positive
12	abounds	positive

```
1 nrc <- get_sentiments("nrc")
2 print(nrc, n = 100)
```

1	abacus	trust
2	abandon	fear
3	abandon	negative
4	abandon	sadness
5	abandoned	anger
6	abandoned	fear
7	abandoned	negative
8	abandoned	sadness
9	abandonment	anger
10	abandonment	fear
11	abandonment	negative
12	abandonment	sadness

A Chinese lexicon [here](#)

Which words reflect “joy” in Jane Austen’s books?

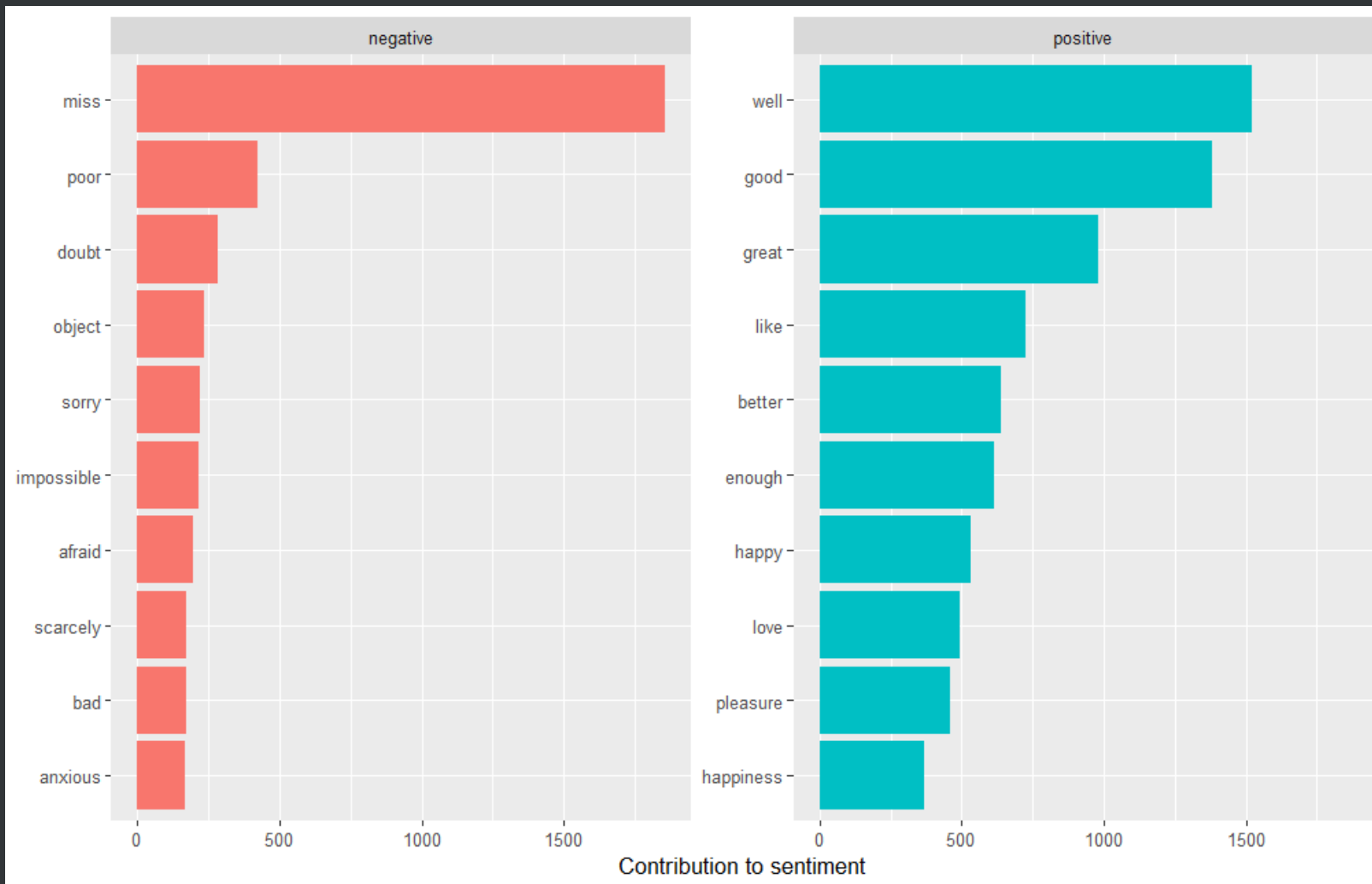
```
1 library(janeaustenr)
2 library(dplyr)
3 library(stringr)
4 tidy_books <- austen_books() %>%
5   unnest_tokens(word, text)
6
7 nrcjoy <- get_sentiments("nrc") %>%
8   filter(sentiment == "joy")
9
10 tidy_books %>%
11   inner_join(nrcjoy) %>%
12   count(word, sort = TRUE)
```

Most common positive/negative words

```
1  bing_word_counts <- tidy_books %>%  
2    inner_join(get_sentiments("bing")) %>%  
3    count(word, sentiment, sort = TRUE)  
4  
5  bing_word_counts
```

Visualizing top positive/negative words

```
1  bing_word_counts %>%
2    group_by(sentiment) %>%
3    slice_max(n, n = 10) %>%
4    mutate(word = reorder(word, n)) %>%
5    ggplot(aes(n, word, fill = sentiment)) +
6    geom_col(show.legend = FALSE) +
7    facet_wrap(~sentiment, scales = "free_y") +
8    labs(x = "Contribution to sentiment",
9         y = NULL)
```



<https://www.youtube.com/embed/si8zZHkufRY?enablejsapi=1>

n-gram

n -gram

In the analysis so far, we've considered words as individual units, and considered their relationships to sentiments. However, 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.

n -gram

Consider the following text: “The University of Hong Kong is the oldest institution in Hong Kong.” In the text, we have tokens such as “University”, “institution”. In addition to that, we have consecutive sequences of words, which are known as n -gram:

- bigram ($n = 2$): Hong Kong, oldest institution, ...
- trigram ($n = 3$): University of Hong Kong, the oldest institution, ...

n-gram

```
1 library(dplyr)
2 library(tidytext)
3 library(janeaustenr)
4 library(tidyr)
5
6
7 austen_bigrams <- austen_books() %>%
8   unnest_tokens(bigram, text, token = "ngrams", n = 2) %>%
9   count(bigram, sort = TRUE) %>%
10  filter(!is.na(bigram))
11
12 austen_bigrams
```

Here, we create bigrams from Jane Austen's books, sort them by frequency, and remove empty bigrams.

n -gram

```
> austen_bigrams
# A tibble: 193,209 x
  bigram      n
  <chr>    <int>
1 of the    2853
2 to be     2670
3 in the    2221
4 it was    1691
5 i am      1485
6 she had   1405
7 of her    1363
8 to the    1315
9 she was   1309
10 had been 1206
```

All of these bigrams are meaningless!

Next Step: Remove bigrams containing stop words.

n-gram

```
1 bigrams_separated <- austen_bigrams %>%
2   separate(bigram, c("word1", "word2"), sep = " ")
3
4 bigrams_filtered <- bigrams_separated %>%
5   filter(!word1 %in% stop_words$word) %>%
6   filter(!word2 %in% stop_words$word)
7
8 bigrams_filtered
```

The complete code is here.

```
1 library(dplyr)
2 library(tidytext)
3 library(janeaustenr)
4 library(tidyr)
5
6
7 austen_bigrams <- austen_books() %>%
8   unnest_tokens(bigram, text, token = "ngrams", n = 2) %>%
9   count(bigram, sort = TRUE) %>%
10  filter(!is.na(bigram))
11
12 bigrams_separated <- austen_bigrams %>%
13   separate(bigram, c("word1", "word2"), sep = " ")
14
15 bigrams_filtered <- bigrams_separated %>%
16   filter(!word1 %in% stop_words$word) %>%
17   filter(!word2 %in% stop_words$word)
18
19 bigrams_filtered
```

How about trigrams?

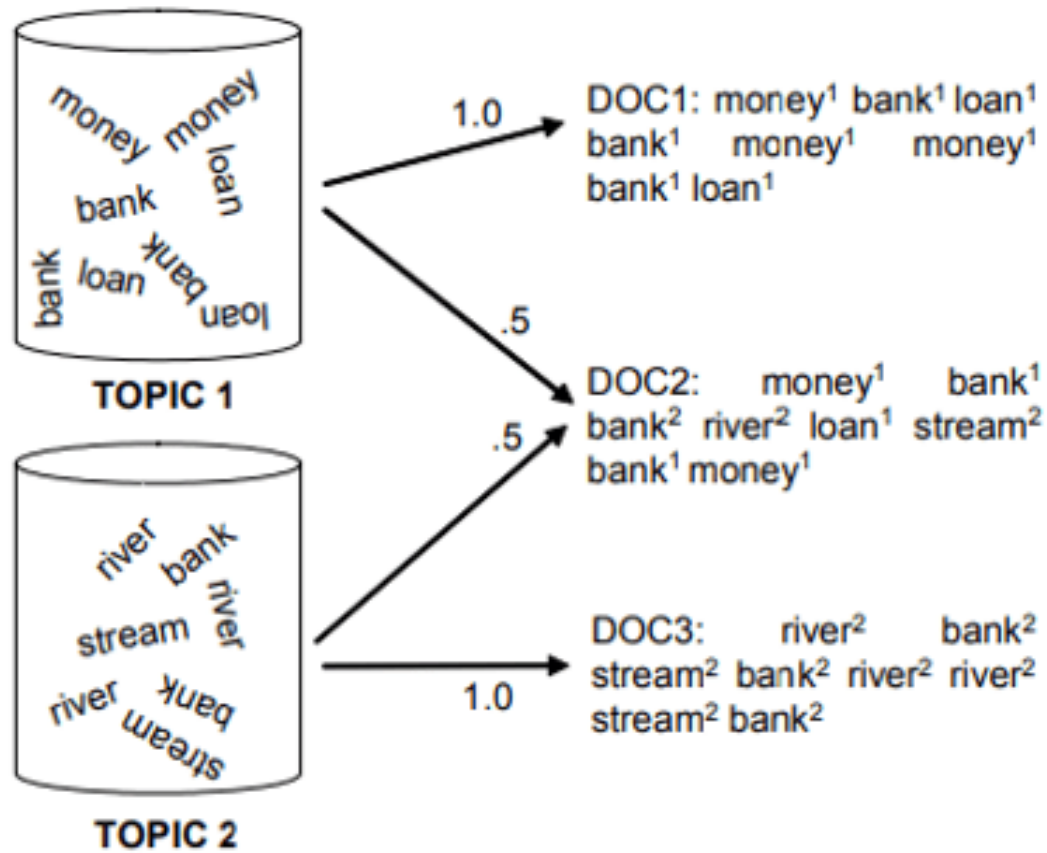
```
1 austen_books() %>%
2   unnest_tokens(trigram, text, token = "ngrams", n = 3) %>%
3   filter(!is.na(trigram)) %>%
4   separate(trigram, c("word1", "word2", "word3"), sep = " ") %>%
5   filter(!word1 %in% stop_words$word,
6          !word2 %in% stop_words$word,
7          !word3 %in% stop_words$word) %>%
8   count(word1, word2, word3, sort = TRUE)
```

Topic Models

Topic Models

In text mining, we often have collections of documents, such as blog posts or news articles, that we'd like to divide into natural groups so that we can understand them separately.

PROBABILISTIC GENERATIVE PROCESS



Topics

gene	0.04
dna	0.02
genetic	0.01
...	

life	0.02
evolve	0.01
organism	0.01
...	

brain	0.04
neuron	0.02
nerve	0.01
...	

data	0.02
number	0.02
computer	0.01
...	

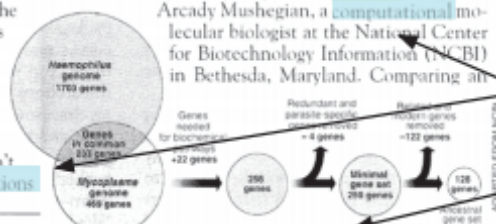
Documents

Seeking Life's Bare (Genetic) Necessities

COLD SPRING HARBOR, NEW YORK—How many **genes** does an **organism** need to **survive**? Last week at the genome meeting here,* two genome researchers with radically different approaches presented complementary views of the basic genes needed for **life**. One research team, using **computer** analyses to compare known **genomes**, concluded that today's **organisms** can be sustained with just 250 genes, and that the earliest life forms required a mere 128 **genes**. The other researcher mapped genes in a simple parasite and estimated that for this organism, 800 genes are plenty to do the job—but that anything short of 100 wouldn't be enough.

Although the numbers don't match precisely, those **predictions**

"are not all that far apart," especially in comparison to the 75,000 **genes** in the human genome, notes Siv Andersson at Uppsala University in Sweden, who arrived at the 800 number. But coming up with a consensus answer may be more than just a **genetic** numbers game, particularly as more and more **genomes** are completely mapped and sequenced. "It may be a way of organizing any newly **sequenced genome**," explains Arcady Mushegian, a **computational** molecular biologist at the National Center for Biotechnology Information (NCBI) in Bethesda, Maryland. Comparing an

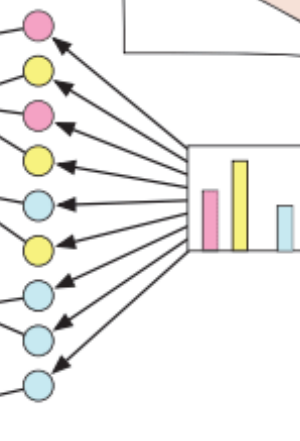


* Genome Mapping and Sequencing, Cold Spring Harbor, New York, May 8 to 12.

Stripping down. **Computer analysis** yields an estimate of the minimum modern and ancient genomes.

SCIENCE • VOL. 272 • 24 MAY 1996

Topic proportions and assignments



Latent Dirichlet Allocation

Latent Dirichlet allocation (LDA) is a famous topic modeling algorithm developed by three computer scientists, David Blei, Andrew Ng and Michael I. Jordan in 2003. Since then, LDA has become one of the most fundamental machine learning algorithms. You can find the original paper for LDA [here](#).

You don't need to understand all the mathematics. We just want to use the algorithm directly.

Latent Dirichlet Allocation

David M. Blei

*Computer Science Division
University of California
Berkeley, CA 94720, USA*

BLEI@CS.BERKELEY.EDU

Andrew Y. Ng

*Computer Science Department
Stanford University
Stanford, CA 94305, USA*

ANG@CS.STANFORD.EDU

Michael I. Jordan

*Computer Science Division and Department of Statistics
University of California
Berkeley, CA 94720, USA*

JORDAN@CS.BERKELEY.EDU

Editor: John Lafferty

Abstract

We describe *latent Dirichlet allocation* (LDA), a generative probabilistic model for collections of discrete data such as text corpora. LDA is a three-level hierarchical Bayesian model, in which each item of a collection is modeled as a finite mixture over an underlying set of topics. Each topic is, in turn, modeled as an infinite mixture over an underlying set of topic probabilities. In the context of text modeling, the topic probabilities provide an explicit representation of a document. We present efficient approximate inference techniques based on variational methods and an EM algorithm for empirical Bayes parameter estimation. We report results in document modeling, text classification, and collaborative filtering, comparing to a mixture of unigrams model and the probabilistic LSI model.

<https://www.youtube.com/embed/p1I9Sa1IRvk?enablejsapi=1>

Demonstration

Please visit [here](https://mimno.infosci.cornell.edu/jsLDA/jslda.html) for an online demonstration of LDA.
(<https://mimno.infosci.cornell.edu/jsLDA/jslda.html>)

The source files are available on the course website.

You can also try the R code on the course website.

Additional Reading (optional):

