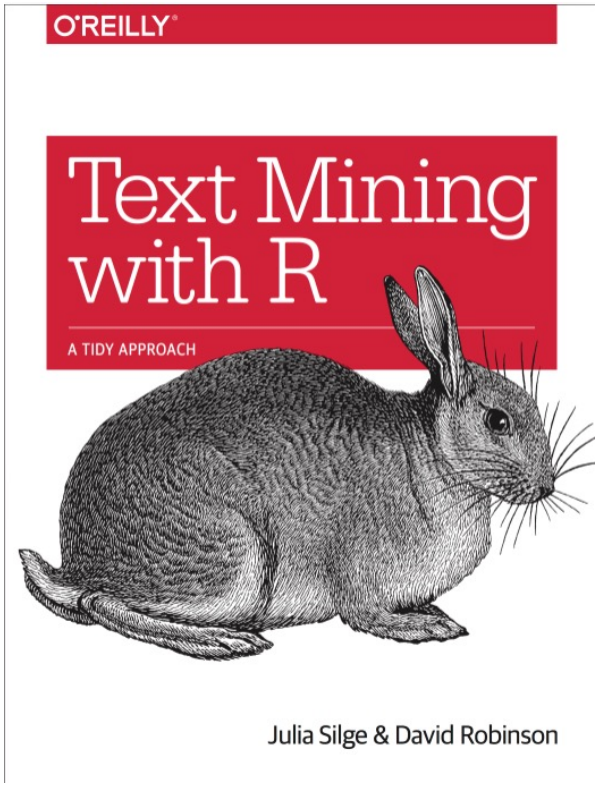
TidyText包

姜冉
陈诗
姜鑫宇
赵晓浩

参考教材

- TidyText的讲解主要基于OREILLY的TEXT Mining with R这本书。



整洁的文本格式

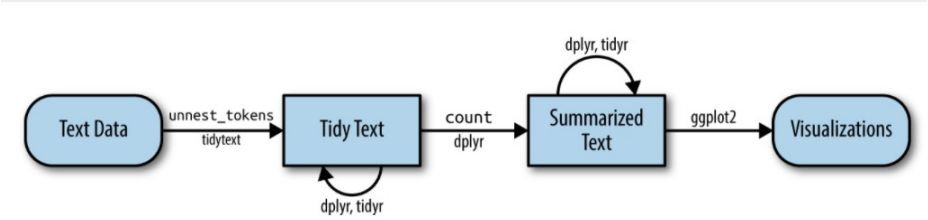
PART.01

Tidy data的特征

- 使用整洁的数据原则，可以使数据处理更容易、更有效，在处理文本时也是这样。
- Hadley Wickham (Wickham 2014) 认为：
- tidy data有一个特定的结构:
  - 每个变量都是一个列
  - 每一个观察值都是一行
  - 每种观测单元都是一个表格

Tidy text包

- tidytext包并不要求在分析过程中始终保持文本数据的整洁形式。
- Tidytext包允许使用dplyr和其他整洁工具完成导入、过滤和处理的工作流，然后将数据转换为机器学习应用程序的文档术语矩阵。然后可以将模型重新转换为一种整洁的形式，以便使用ggplot2进行解释和可视化。



Tidy text的数据结构

01

字符串

即字符向量，通常首先以这种形式将文本数据读入内存。

02

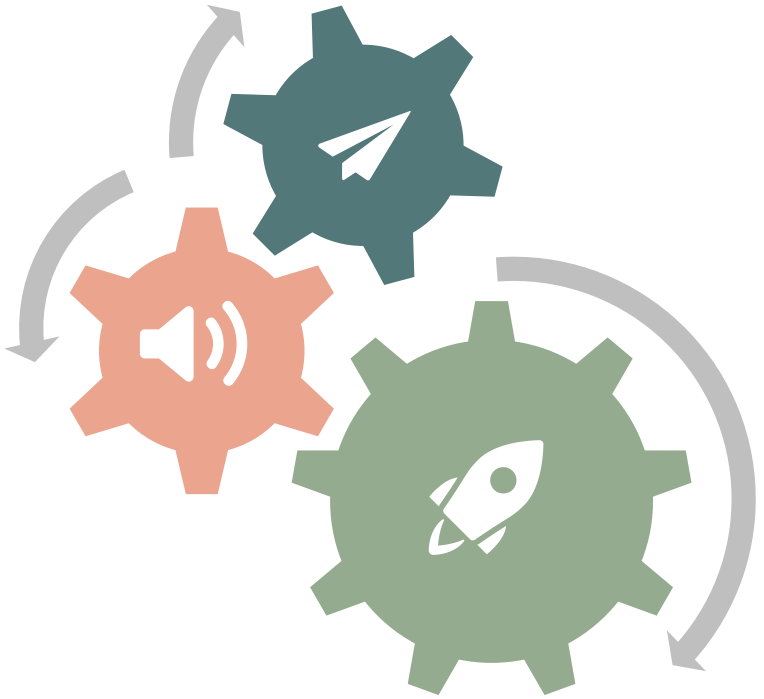
语料库：

这些类型的对象通常包含带有附加元数据和详细信息注释的原始字符串。

03

文档术语矩阵：

这是一个稀疏矩阵，描述文档的集合（即语料库），每个文档一行，每个术语一列。矩阵中的值通常是字数或tf-idf。



token

- 整洁的文本格式：一行一个“token（标记）”的表
- Token是用于分析的有意义的文本单元（例如单词），token化是将文本拆分为一个个token的过程。
- 对于整洁化文本挖掘，存储在每一行中的**token**通常是单个单词，但也可以是句子或段落。
- 在tidytext包中，提供了按此类常用文本单位进行token化，并转换为每行一个词的功能。

unnest_tokens函数

- 功能：将文本分解成单独的token（称为token化的过程），然后将其转换为整洁的数据结构。
- 例子：

```
text <- c("Because I could not stop for Death -",
          "He kindly stopped for me -",
          "The Carriage held but just Ourselves -",
          "and Immortality")

text
```

```
#> [1] "Because I could not stop for
Death -"
#> [2] "He kindly stopped for me -"
#> [3] "The Carriage held but just
Ourselves -"
#> [4] "and Immortality"
```

```
library(dplyr)
text_df <- tibble(line = 1:4, text =
text)

text_df
```

```
#> # A tibble: 4 x 2
#>    line text
#>   <int> <chr>
#> 1     1 Because I could not stop for Death -
#> 2     2 He kindly stopped for me -
#> 3     3 The Carriage held but just Ourselves -
#> 4     4 and Immortality
```

tibble是R中的一种现代数据帧，可在dplyr和tibble软件包中使用
它不会将字符串转换为因子，并且不使用行名。
非常适合与tidy text配合使用。

但是此时包含文本的数据框无法过滤掉最频繁出现的单词或计数，因为每一行都是由多个组合单词组成的。
我们需要对其进行转换，以使其每行每个文档具有一个token。

```
library(tidytext)

text_df %>%
  unnest_tokens(word, text)
```

这里使用的unnest_tokens的两个基本参数是列名。
首先，我们有输出列名(在本例中是word)，然后是文本来自的输入列(在本例中是text)

在使用unnest_tokens之后，我们分割了每一行，以便在新数据帧的每一行中都有一个token;

```
#> # A tibble: 20 x 2
#>    line word
#>   <int> <chr>
#>  1     1 because
#>  2     1 i
#>  3     1 could
#>  4     1 not
#>  5     1 stop
#>  6     1 for
#>  7     1 death
#>  8     2 he
#>  9     2 kindly
#> 10     2 stopped
#> # … with 10 more rows
```

注意:

其他列，如每个单词的行号被保留。
标点被剥离。
默认情况下，unnest_tokens()将token转换为小写，使得它们更容易与其他数据集进行比较或组合。(使用to_lower = FALSE参数可以关闭自动小写)。

字频

- 文本挖掘中的一个常见任务是，查看单词的频率，并比较不同文本的频率。
- 以简·奥斯丁的小说作为例子：

```
tidy_bronte %>%
  count(word, sort = TRUE)
```

```
#> # A tibble: 23,051 x 2
#>    word       n
#>    <chr>  <int>
#>  1 time    1065
#>  2 miss     855
#>  3 day      827
#>  4 hand     768
#>  5 eyes     713
#>  6 night    647
#>  7 heart    638
#>  8 looked   602
#>  9 door     592
#> 10 half     586
#> # ... with 23,041 more rows
```

Tidy Data的情感分析

PART.02

情感分析

- Tidy Text可以处理文本的情感内容。

  --当读者阅读文本时，我们会根据对词的情感意图的理解来推断一段文本是正面的还是负面的，或者以某种其他更细微的情感来作为特征，例如惊奇或厌恶。

- 分析方法：

  --分析文本情感的一种方法是将文本视为其单个单词的组合，并将整个文本的情感内容视为单个单词的情感内容的总和。这不是进行情感分析的唯一方法，但它是一种常用的方法，*并且*自然地利用了整洁的工具生态系统。

情感词典

- tidytext包提供对多个情感词典的访问：
  - AFINN from Finn Årup Nielsen,
  - bing from Bing Liu and collaborators,
  - nrc from Saif Mohammad and Peter Turney.
- 这三个词典都基于单字组，即单个单词。
- 词典中包含许多英语单词，并为单词分配积极/消极情绪的分数，还可能赋予诸如喜悦，愤怒，悲伤等情绪的分数。

get_sentiments()

- 获得针对特定情感词典的词汇，并针对每个词汇采取适当的措施。

```
> get_sentiments("bing")
# A tibble: 6,786 x 2
   word         sentiment
   <chr>        <chr>
 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
# ... with 6,776 more rows
> |
```

```
get_sentiments("nrc")
```

```
#> # A tibble: 13,901 x 2
#>    word        sentiment
#>    <chr>       <chr>
#>  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
#> # … with 13,891 more rows
```

```
get_sentiments("afinn")
```

```
#> # A tibble: 2,477 x 2
#>    word       value
#>    <chr>      <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
#> # … with 2,467 more rows
```

注意：

- 并不是每个英语单词都在情感词典中。
  - 因为许多英语单词是中性的。而且这些方法不考虑单词前的限定词，例如“不好”或“不正确”。像这样的基于词典的方法仅基于字母组合。
- 用来加总单字情感分数的文本块的大小可能会对分析产生影响。许多段落大小的文本通常可以将正负情绪平均为零。

- 使用NRC词典和filter()。
- 让filter()数据框包含书中的文本，以表示来自Emma的单词
- inner_join()执行情感分析。
- count()：最常见的喜乐词。

```
nrc_joy <- get_sentiments("nrc") %>%
  filter(sentiment == "joy")

tidy_books %>%
  filter(book == "Emma") %>%
  inner_join(nrc_joy) %>%
  count(word, sort = TRUE)
```

```
#> # A tibble: 303 x 2
#>    word        n
#>    <chr>   <int>
#>  1 good      359
#>  2 young     192
#>  3 friend    166
#>  4 hope      143
#>  5 happy     125
#>  6 love      117
#>  7 deal       92
#>  8 found      92
#>  9 present    89
#> 10 kind       82
#> # … with 293 more rows
```

分析词与文档的频率

PART.03

第三部分分析词与文档的频率：tf-idf

tf-idf是一种用于信息检索与数据挖掘的常用加权技术。用以评估一字词对于一个文件集或一个语料库中的其中一份文件的重要程度。字词的重要性随着它在文件中出现的次数成正比增加，但同时会随着它在语料库中出现的频率成反比下降。该技术包括两个部分：

1词频 term-frequency（tf）
2 逆文本频率指数 inverse document frequency（idf）

1词频 term-frequency

n/total：n为该词在文档中出现的次数
total为文档中总词数

以简奥斯汀的小说为例（janeaustenr包）

A tibble: 40,379 x 4

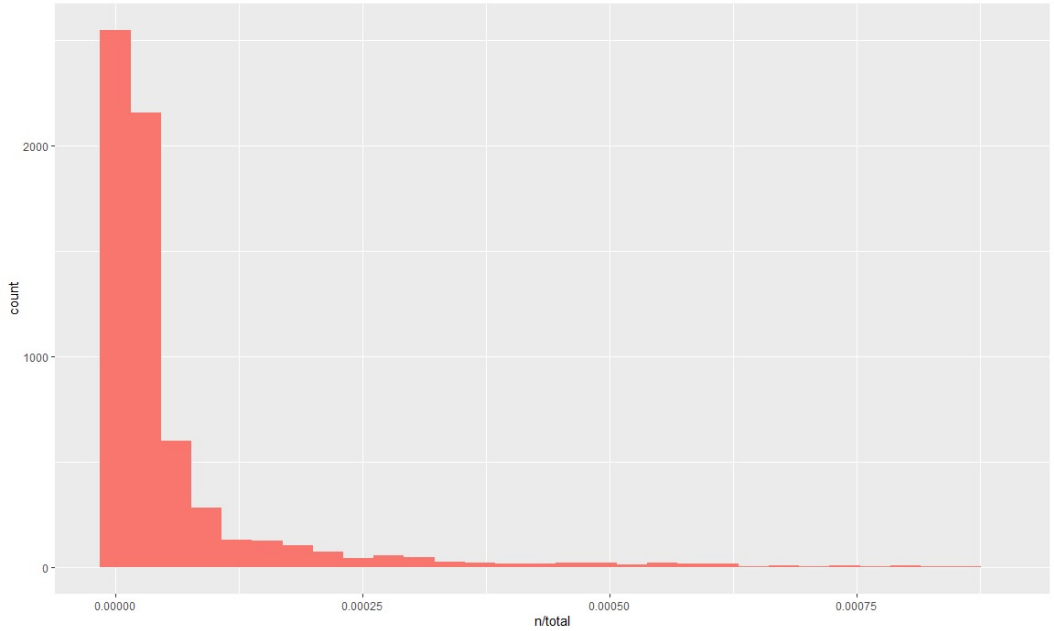
	book	word	n	total
	<fct>	<chr>	<int>	<int>
1	Mansfield Park	the	6206	160460
2	Mansfield Park	to	5475	160460
3	Mansfield Park	and	5438	160460
4	Emma	to	5239	160996
5	Emma	the	5201	160996
6	Emma	and	4896	160996
7	Mansfield Park	of	4778	160460
8	Pride & Prejudice	the	4331	122204
9	Emma	of	4291	160996
10	Pride & Prejudice	to	4162	122204

... with 40,369 more rows

```
> book_words <- austen_books() %>%
+   unnest_tokens(word, text) %>%
+   count(book, word, sort = TRUE)
> total_words <- book_words %>%
+   group_by(book) %>%
+   summarize(total = sum(n))
> book_words <- left_join(book_words, total_words)
```

1词频 term-frequency

n/total：n为该词在文档中出现的次数
total为文档中总词数
统计Pride & prejudice中的词频：




```
PP_books <- book_words %>%
  filter(book =="Pride & Prejudice")
gplot(PP_books, aes(n/total, fill = book)) +
    geom_histogram(show.legend = FALSE) +
    xlim(NA, 0.0009)
```

2 tf-idf

逆文本频率指数：
由总文件数目除以包含该词语之文件的数目，再将得到的商取以10为底的对数得到。

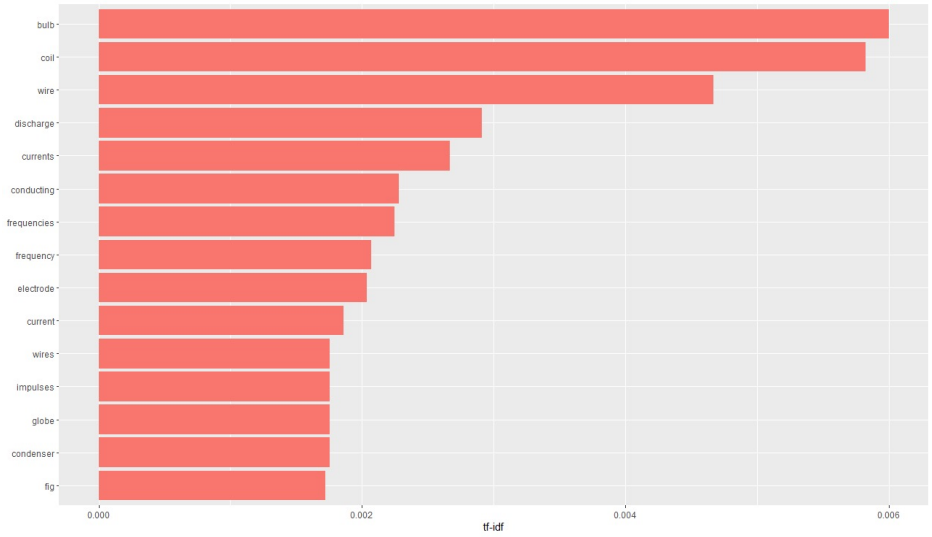
$$idf(\text{term}) = \ln\left(\frac{n_{\text{documents}}}{n_{\text{documents containing term}}}\right)$$

2 tf-idf

bind_tf_idf(tbl, term, document,

Arguments

`tbl`	A tidy text dataset with one-row-per-term-per-document
`term`	Column containing terms as string or symbol
`document`	Column containing document IDs as string or symbol
`n`	Column containing document-term counts as string or symbol




```
plot_physics <- physics_words %>%
  bind_tf_idf(word, author, n) %>%
  mutate(author = factor(author, levels = c("Galilei, Galileo",
                                            "Huygens, Christiaan",
                                            "Tesla, Nikola",
                                            "Einstein, Albert")))
plot_TN<-plot_physics %>%
  filter(author == "Tesla, Nikola") %>%
```

对Tesla, Nikola文章的td_idf处理结果可视化

n-gram & correlations

PART.04

第四部分n-gram & correlations

n-gram是一种基于统计语言模型的算法。它的基本思想是将文本里面的内容按照字节进行大小为N的滑动窗口操作，形成了长度是N的字节片段序列。

常用的是二元的Bi-Gram和三元的Tri-Gram

Bigram

对janaustenr包中的《Pride&prejudice》进行分词处理，将unnest_tokens函数中的token参数设置为ngram

```
library(dplyr)
library(tidytext)
library(janeaustenr)

austen_bigrams <- austen_books() %>%
  filter(book == "Pride & Prejudice") %>%
  unnest_tokens(bigram, text, token = "ngrams", n = 2)

austen_bigrams
```

```
book                bigram
<fct>               <chr>
Pride & Prejudice pride and
Pride & Prejudice and prejudice
Pride & Prejudice NA
Pride & Prejudice by jane
Pride & Prejudice jane austen
Pride & Prejudice NA
Pride & Prejudice NA
Pride & Prejudice NA
Pride & Prejudice chapter 1
Pride & Prejudice NA
... with 114,035 more rows
```

Bigram

```
> austen_bigrams %>%
+   count(bigram, sort = TRUE)
# A tibble: 50,266 x 2
   bigram       n
   <chr>    <int>
 1 NA        2556
 2 of the     439
 3 to be      422
 4 in the     365
 5 i am       291
 6 of her     245
 7 it was     235
 8 to the     231
 9 mr darcy   230
10 of his     219
# ... with 50,256 more rows
> |
```

Bigram

stop words指在不具有实际意义的单词，一般包括of in on等，为了对文本进行更有效的分析，我们需要避免stopwords的影响。
在上一步bigram的基础上进行分词，筛选出stop words

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

	word1	word2	n
	<chr>	<chr>	<int>
1	NA	NA	2556
2	lady	catherine	87
3	miss	bingley	67
4	miss	bennet	52
5	sir	william	35
6	de	bourgh	32
7	miss	darcy	32
8	cried	elizabeth	24
9	colonel	forster	23
10	miss	lucas	23

... with 5,099 more rows

Correlations

在之前对单个词语分析的基础上，分析多个词语之间的关系。

```
library(igraph)
1分词处理
austen_bigrams <- austen_books() %>%
  unnest_tokens(bigram, text, token = "ngrams", n = 2)
austen_bigrams %>%
  count(bigram, sort = TRUE)
```

```
> bigram_graph
IGRAPH e232022 DN-- 86 71 --
+ attr: name (v/c), n (e/n)
+ edges from e232022 (vertex names):
 [1] NA       ->NA         sir      ->thomas     miss     ->crawford
 [4] captain ->wentworth miss     ->woodhouse frank    ->churchill
 [7] lady     ->russell    sir      ->walter     lady     ->bertram
[10] miss     ->fairfax    colonel ->brandon    sir      ->john
[13] miss     ->bates      jane     ->fairfax    lady     ->catherine
[16] lady     ->middleton miss     ->tilney     miss     ->bingley
[19] thousand->pounds     miss     ->dashwood   dear     ->miss
[22] miss     ->bennet     miss     ->morland    captain ->benwick
+ ... omitted several edges
>
```

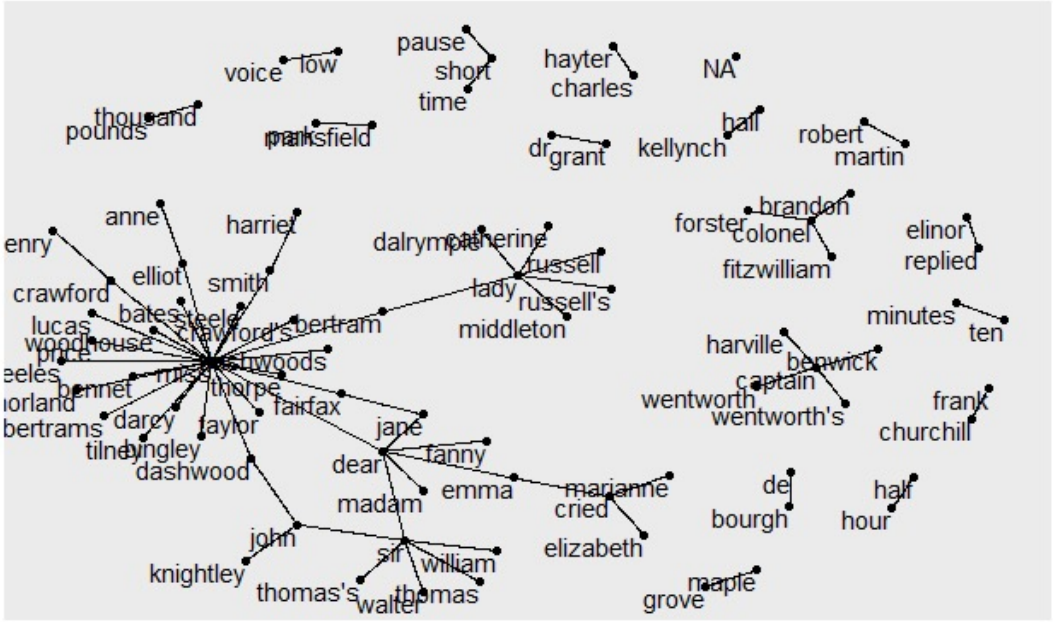
2筛选出现频率高的单词，并使用igraph包中的graph_from_data_frame()将分析对象转变为便于分析的数据框格式。

```
bigram_graph <- bigram_counts %>%
  filter(n > 20) %>%
  graph_from_data_frame()
```

Correlations

在之前对单个词语分析的基础上，分析多个词语之间的关系，实现结果可视化。

```
library(ggraph)
set.seed(2017)//设定随机数种子
ggraph(bigram_graph, layout = "fr") +
  geom_edge_link() +
  geom_node_point() +
  geom_node_text(aes(label = name),
                 vjust = 1, hjust = 1)
```



转化为非整洁格式的方法

PART.05

文档术语矩阵

- 文本挖掘程序包可使用的最常见的结构之一是文档术语矩阵（或DTM）。这是一个矩阵，其中：
- 每行代表一个文档（例如书或文章），
- 每列代表一个术语，并且
- 每个值（通常）包含该术语在该文档中的出现次数。

- DTM对象不能直接与整洁的工具一起使用，就像整洁的数据帧不能用作大多数文本挖掘程序包的输入一样。因此，tidytext包提供了两个在两种格式之间转换的动词。
- tidy()将文档项矩阵转换为整洁的数据帧。
- cast()将整洁的每行一期数据帧转换为矩阵。

```
ap_td <- tidy(AssociatedPress)
ap_td
#> # A tibble: 302,031 x 3
#>    document term       count
#>       <int> <chr>      <dbl>
#>  1        1 adding         1
#>  2        1 adult          2
#>  3        1 ago            1
#>  4        1 alcohol        1
#>  5        1 allegedly      1
#>  6        1 allen          1
#>  7        1 apparently     2
#>  8        1 appeared       1
#>  9        1 arrested       1
#> 10        1 assault        1
#> # … with 302,021 more rows
```

tidy()动词，该动词采用一个非整洁的对象并将其转换为整洁的数据帧

Cast()将整洁的文本数据转换为矩阵

```
ap_td %>%
  cast_dtm(document, term, count)
#> <<DocumentTermMatrix (documents: 2246, terms: 10473)>>
#> Non-/sparse entries: 302031/23220327
#> Sparsity           : 99%
#> Maximal term length: 18
#> Weighting          : term frequency (tf)
```

整洁的文本提供了cast_用于从整洁的形式转换为这些矩阵的动词。
我们可以使用整理后的AP数据集并将其转换回文档项矩阵cast_dtm()

主题建模

PART.06

6.主题建模

潜在狄利克雷分配

- 潜在Dirichlet分配是用于主题建模的最常见算法之一。无需深入研究模型背后的数学原理，我们就可以将其理解为受两个原理指导。
- **每个文档都是主题的混合体。**
- **每个主题都是单词的混合体。**。
- LDA是一种用于同时估计这两者的数学方法：查找与每个主题相关的单词的混合，同时还确定描述每个文档的主题的混合。

单词主题概率

```
library(tidytext)

ap_topics <- tidy(ap_lda, matrix = "beta")
ap_topics
#> # A tibble: 20,946 x 3
#>    topic term           beta
#>    <int> <chr>         <dbl>
#>  1     1 aaron      1.69e-12
#>  2     2 aaron      3.90e- 5
#>  3     1 abandon    2.65e- 5
#>  4     2 abandon    3.99e- 5
#>  5     1 abandoned  1.39e- 4
#>  6     2 abandoned  5.88e- 5
#>  7     1 abandoning 2.45e-33
#>  8     2 abandoning 2.34e- 5
#>  9     1 abbott     2.13e- 6
#> 10     2 abbott     2.97e- 5
#> # … with 20,936 more rows
```

Tidy()用于从模型中提取每个主题每个单词的概率，称为β对于每种组合，模型都会计算从该主题生成该术语的概率。

文档主题概率

```
ap_documents <- tidy(ap_lda, matrix = "gamma")
ap_documents
#> # A tibble: 4,492 x 3
#>    document topic    gamma
#>       <int> <int>    <dbl>
#>  1        1     1 0.248
#>  2        2     1 0.362
#>  3        3     1 0.527
#>  4        4     1 0.357
#>  5        5     1 0.181
#>  6        6     1 0.000588
#>  7        7     1 0.773
#>  8        8     1 0.00445
#>  9        9     1 0.967
#> 10       10     1 0.147
#> # … with 4,482 more rows
```

Tidy（）检查每个文档每话题的概率，被称为伽玛。

这些值中的每一个都是该文档中从该主题生成的单词的估计比例。例如，该模型估计文档1中仅约25%的单词是从主题1生成的。

主题建模例子

- 假设一个破坏者闯入了您的书房，并撕碎了您的四本书：
- *远大前程*狄更斯
- HG威尔斯*的《世界大战》*
- *海底两万个联盟*（Jules Verne）
- 简·奥斯丁的《*傲慢与偏见*》
- 这个破坏者将这些书撕成单独的章节，并将它们堆成一堆。我们如何将这些杂乱无章的章节还原为原始书籍？这是一个具有挑战性的问题，因为各章**未标记**：我们不知道哪些词会将它们分为几类。因此，我们将使用主题建模来发现各章如何组合成不同的主题，每个主题（大概）代表一本书。
- 我们将使用第3章中介绍的gutenbergr包来检索这四本书的文本。

```
titles <- c("Twenty Thousand Leagues under the Sea",
            "The War of the Worlds",
            "Pride and Prejudice",
            "Great Expectations")
```

```
library(gutenbergr)

books <- gutenberg_works(title %in% titles) %>%
  gutenberg_download(meta_fields = "title")
```

作为预处理，我们将它们分为不同的章节，使用整洁的`unnest_tokens()`文字将它们分成单词，然后删除`stop_words`。我们将每一章都视为一个单独的“文档”，每个名称都使用`Great Expectations_1`或`Pride and Prejudice_11`。（在其他应用程序中，每个文档可能是一篇报纸文章或一篇博客文章）。

```
library(stringr)

# divide into documents, each representing one chapter
by_chapter <- books %>%
  group_by(title) %>%
  mutate(chapter = cumsum(str_detect(
    text, regex("^chapter ", ignore_case = TRUE)
  ))) %>%
  ungroup() %>%
  filter(chapter > 0) %>%
  unite(document, title, chapter)
```

现在，我们的数据框架`word_counts`是整齐的格式，每个文档每行一个词，但是topicmodels包需要一个`DocumentTermMatrix`。如第5.2节所述，我们可以将每行一个令牌表转换为`DocumentTermMatrix`带有tidytext的表`cast_dtm()`。

```
chapters_dtm <- word_counts %>%
  cast_dtm(document, word, n)

chapters_dtm
#> <<DocumentTermMatrix (documents: 193, terms: 18215)>>
#> Non-/sparse entries: 104721/3410774
#> Sparsity           : 97%
#> Maximal term length: 19
#> Weighting          : term frequency (tf)
```

然后，我们可以使用该 `LDA()` 函数创建一个四主题模型。在这种情况下，我们知道我们正在寻找四个主题，因为有四本书。在其他问题中，我们可能需要尝试一些不同的值 `k`。

```
chapters_lda <- LDA(chapters_dtm, k = 4, control = list(seed = 1234))
chapters_lda
#> A LDA_VEM topic model with 4 topics.
```

我们可以使用dplyr `slice_max()` 查找每个主题中的前5个字词。

```
top_terms <- chapter_topics %>%
  group_by(topic) %>%
  slice_max(beta, n = 5) %>%
  ungroup() %>%
  arrange(topic, -beta)
```

这个整洁的输出非常适合ggplot2可视化

这些主题显然与这四本书相关联！毫无疑问，“船长”，“鹦鹉螺”，“海”和“尼莫”的主题属于*海底两万个同盟*，而“吉恩”，“达西”和“伊丽莎白”则属于*“骄傲”和偏见*。我们从*《远大前程》*中看到“点子”和“乔伊”，从*《世界大战》*中看到“mart夫”，“黑人”和“夜”。我们还注意到，与LDA是一种“模糊聚类”方法相一致，多个主题之间可能会有共同的词，例如主题1和4中的“未命中”以及主题3和4中的“时间”。

案例

PART.07

- 使用整洁的数据原理可以使许多文本挖掘任务更容易，更有效，并且与已经广泛使用的工具保持一致。使用整洁的数据帧进行文本挖掘所需的许多基础结构已经存在于dplyr，broom，tidyr和ggplot2之类的软件包中。在此程序包中，我们提供功能和支持的数据集，以允许将文本与整齐的格式相互转换，并在整齐的工具和现有的文本挖掘程序包之间无缝切换。

```
library(janeaustenr)
library(dplyr)
library(stringr)

original_books <- austen_books() %>%
  group_by(book) %>%
  mutate(line = row_number(),
         chapter = cumsum(str_detect(text, regex("^chapter [\\divxlc]",
                                                 ignore_case = TRUE)))) %>%
  ungroup()

original_books
```

```
## # A tibble: 73,422 x 4
##    text                    book                line chapter
##    <chr>                   <fct>              <int>   <int>
##  1 "SENSE AND SENSIBILITY" Sense & Sensibility     1       0
##  2 ""                      Sense & Sensibility     2       0
##  3 "by Jane Austen"        Sense & Sensibility     3       0
##  4 ""                      Sense & Sensibility     4       0
##  5 "(1811)"                Sense & Sensibility     5       0
##  6 ""                      Sense & Sensibility     6       0
##  7 ""                      Sense & Sensibility     7       0
##  8 ""                      Sense & Sensibility     8       0
##  9 ""                      Sense & Sensibility     9       0
## 10 "CHAPTER 1"             Sense & Sensibility    10       1
## # … with 73,412 more rows
```

简·奥斯丁的小说可以这么整洁！让我们使用janeaustenr软件包中Jane Austen的6篇已完成的已出版小说的文本，并将其转换为整齐的格式。janeaustenr以一行一行的格式提供它们：

要将其作为整洁的数据集使用，我们需要将其重构每行一令牌的格式。该unnest_tokens函数是一种将带有文本列的数据帧转换为每行一令牌的方法：

```
library(tidytext)
tidy_books <- original_books %>%
  unnest_tokens(word, text)

tidy_books
```

```
## # A tibble: 725,055 x 4
##    book                line chapter word
##    <fct>              <int>   <int> <chr>
##  1 Sense & Sensibility     1       0 sense
##  2 Sense & Sensibility     1       0 and
##  3 Sense & Sensibility     1       0 sensibility
##  4 Sense & Sensibility     3       0 by
##  5 Sense & Sensibility     3       0 jane
##  6 Sense & Sensibility     3       0 austen
##  7 Sense & Sensibility     5       0 1811
##  8 Sense & Sensibility    10       1 chapter
##  9 Sense & Sensibility    10       1 1
## 10 Sense & Sensibility    13       1 the
## # … with 725,045 more rows
```

此函数使用tokenizers包将每行分成单词。默认的分词是针对单词的，但是其他选项包括字符，ngram，句子，行，段落或正则表达式模式周围的分隔符。现在数据是每行一个字的格式，我们可以使用诸如dplyr之类的整洁工具来操纵它。我们可以使用来删除停用词（可以使用该函数以整齐的形式访问get_stopwords()）anti_join。

```
cleaned_books <- tidy_books %>%
  anti_join(get_stopwords())
```

我们还可以count用来在整个书籍中找到最常见的单词。

```
cleaned_books %>%
  count(word, sort = TRUE)
```

情感分析可以作为内部联接来完成。情感词典可以通过该get_sentiments()功能获得。让我们看一下刘兵和合作者词典中得分为正的单词。在Emma中最常见的肯定词是什么？

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

tidy_books %>%
  filter(book == "Emma") %>%
  semi_join(positive) %>%
  count(word, sort = TRUE)
```

```
## # A tibble: 668 x 2
##    word         n
##    <chr>    <int>
##  1 well       401
##  2 good       359
##  3 great      264
##  4 like       200
##  5 better     173
##  6 enough     129
##  7 happy      125
##  8 love       117
##  9 pleasure   115
## 10 right       92
## # … with 658 more rows
```

或者，我们可以研究每部小说中的情感变化。让我们使用相同的词典找到每个单词的情感评分，然后计算每本小说中定义部分中正词和负词的数量。

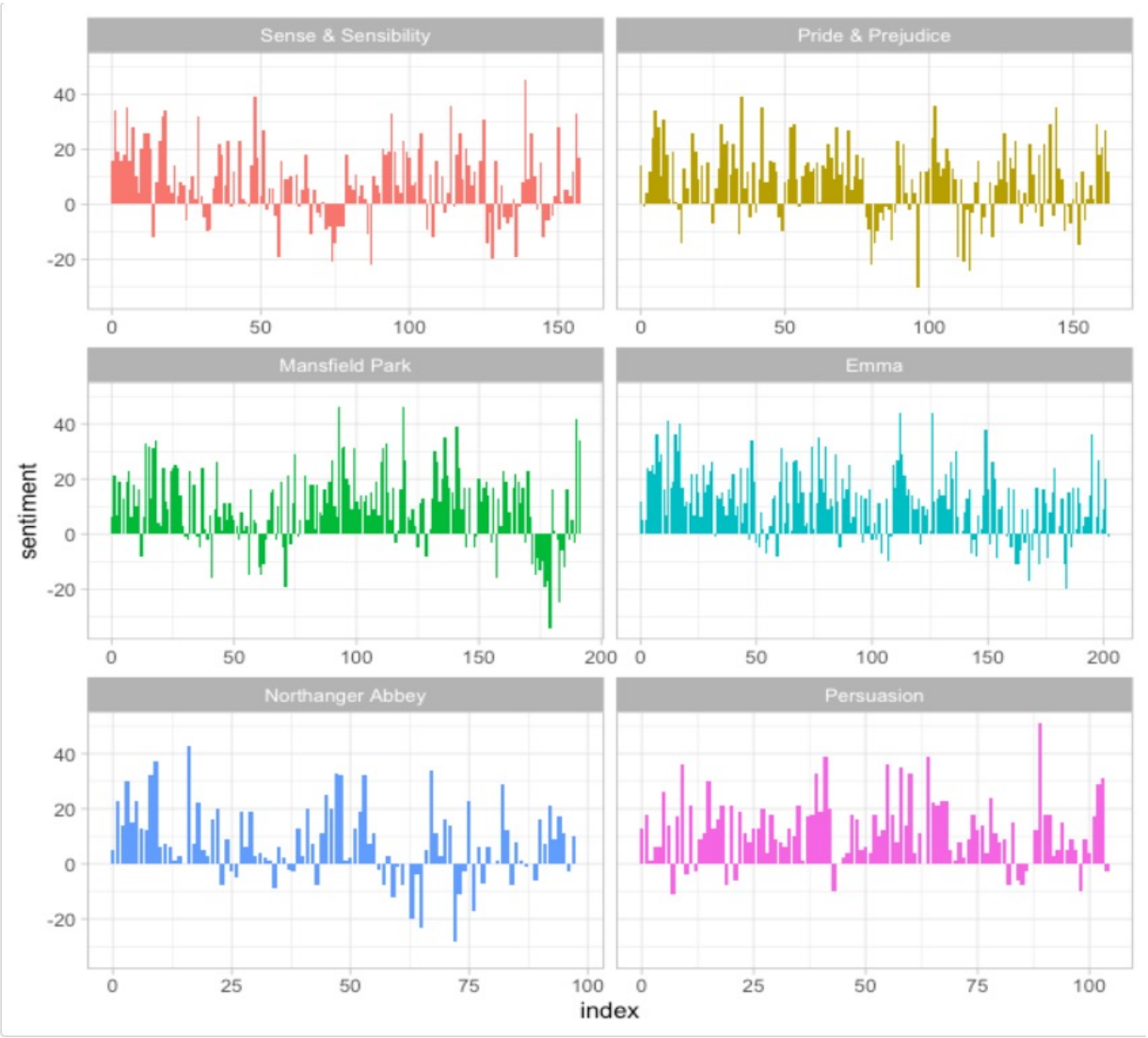
```
library(tidyr)
bing <- get_sentiments("bing")

janeaustensentiment <- tidy_books %>%
  inner_join(bing) %>%
  count(book, index = line %/% 80, sentiment) %>%
  spread(sentiment, n, fill = 0) %>%
  mutate(sentiment = positive - negative)
```

现在我们可以在每本小说的情节轨迹上绘制这些情感分数。

```
library(ggplot2)

ggplot(janeaustensentiment, aes(index, sentiment, fill = book)) +
  geom_bar(stat = "identity", show.legend = FALSE) +
  facet_wrap(~book, ncol = 2, scales = "free_x")
```



该同时具有情感和单词的数据帧的一个优点是，我们可以分析对每个情感都有影响的单词计数。

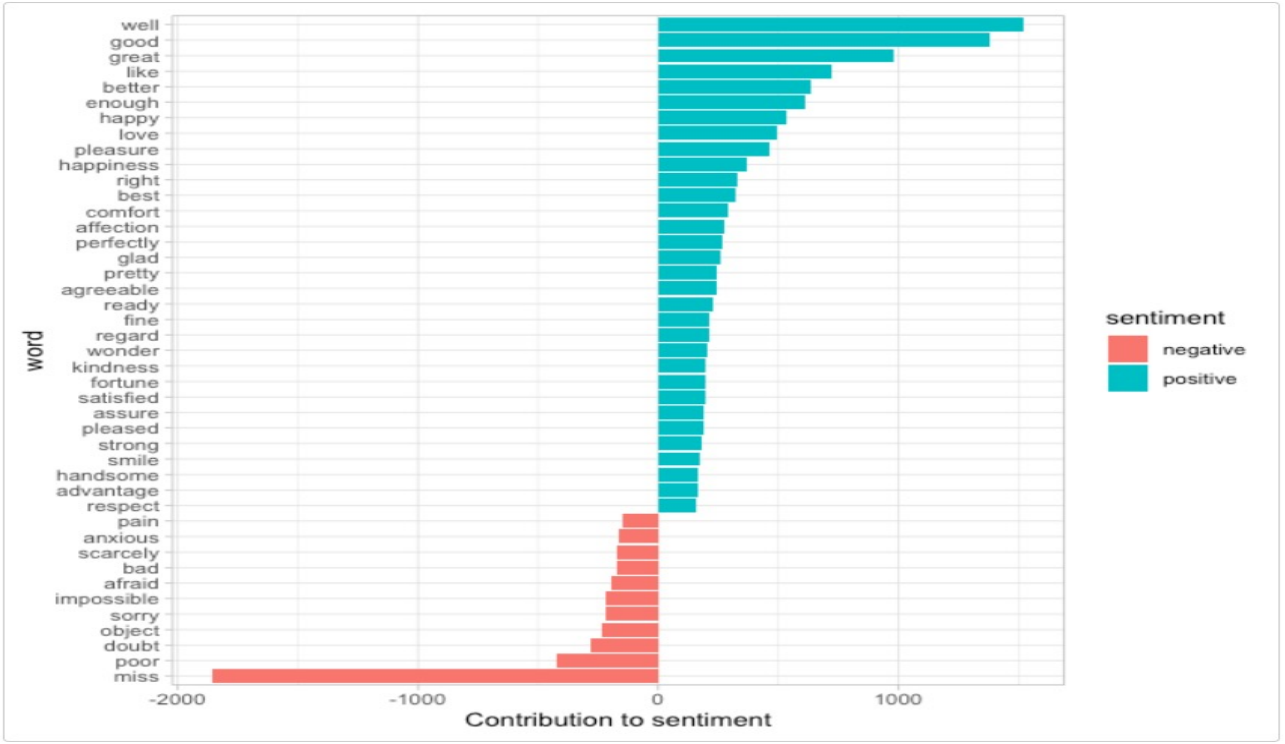
```
bing_word_counts <- tidy_books %>%
  inner_join(bing) %>%
  count(word, sentiment, sort = TRUE)

bing_word_counts
```

```
## # A tibble: 2,585 x 3
##    word     sentiment     n
##    <chr>    <chr>     <int>
##  1 miss     negative   1855
##  2 well     positive   1523
##  3 good     positive   1380
##  4 great    positive    981
##  5 like     positive    725
##  6 better   positive    639
##  7 enough   positive    613
##  8 happy    positive    534
##  9 love     positive    495
## 10 pleasure positive    462
## # … with 2,575 more rows
```

```
bing_word_counts %>%
  filter(n > 150) %>%
  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")
```

这可以直观地显示出来，并且由于可以始终如一地使用为处理整洁的数据帧而构建的工具，因此我们可以直接将其传送到ggplot2中。



我们已经看到，这种整洁的文本挖掘方法可以与ggplot2一起很好地工作，但是以整洁的格式存储数据对于其他绘图也很有用。
例如，考虑wordcloud软件包。让我们再一次看一下简·奥斯丁作品中最常见的词。

```
library(wordcloud)

cleaned_books %>%
  count(word) %>%
  with(wordcloud(word, n, max.words = 100))
```

在其他函数中，例如comparison.cloud，您可能需要将其转换为带有reshape2的矩阵acast。让我们进行情感分析，以使用内部联接标记正面和负面词，然后找到最常见的正面和负面词。直到我们需要将数据发送到的步骤之前comparison.cloud，所有这些操作都可以通过连接，管道和dplyr完成，因为我们的数据是整齐的格式。

```
library(reshape2)

tidy_books %>%
  inner_join(bing) %>%
  count(word, sentiment, sort = TRUE) %>%
  acast(word ~ sentiment, value.var = "n", fill = 0) %>%
  comparison.cloud(colors = c("#F8766D", "#00BFC4"),
                   max.words = 100)
```