文本分析笔记

jieba 分词包

黄蒙

2020-01-08

目录

1 西文文本分析

1.1 分词 unnest_tokens()

tidytext::unnest_tokens(tbl, output, input, token = “words”, format = c(“text”, “man”, “latex”, “html”, “xml”), to_lower = TRUE, drop = TRUE, collapse = NULL, …)

第一个参数为数据框，第二个参数为转换后的列名，第三个参数为被转换的列名。

例：

```
text_df <- tibble(line = 1:4, text = c('I have a dream', 'I want to have a look', 'I lo
text_df
```

```
# A tibble: 4 x 2
   line text
  <int> <chr>
1     1 I have a dream
2     2 I want to have a look
3     3 I love you
4     4 I will go
```

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

```
# A tibble: 16 x 2
    line word
   <int> <chr>
 1     1 i
 2     1 have
 3     1 a
 4     1 dream
 5     2 i
 6     2 want
 7     2 to
```

```
 8     2 have
 9     2 a
10     2 look
11     3 i
12     3 love
13     3 you
14     4 i
15     4 will
16     4 go
```

效果：1. 其他列会被保留 2. 标点符号自动删除 3. to_lower 参数默认将词转换为小写

1.2 词频及词频的跨文档比较

```
library(janeaustenr) # 该包中的数据框存有六本小说，一个观测为一行文本

## 读入文本数据，添加行号和章节
original_books <- austen_books() %>%
  group_by(book) %>%
  mutate(linenumber = row_number(), # 添加行编号
         chapter = cumsum( # 添加章节号
           str_detect(text, regex("^chapter [\\divxlc]",
                                  ignore_case = TRUE))
           )
         ) %>%
  ungroup()
original_books
```

```
# A tibble: 73,422 x 4
   text                  book                linenumber 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
```

```
## 分词，化为一个观测一词的整洁数据
tidy_books <- original_books %>% unnest_tokens(word, text)
tidy_books
```

```
# A tibble: 725,055 x 4
   book                linenumber 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
```

```
## 删除停用词
data(stop_words)
stop_words
```

```
# A tibble: 1,149 x 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
# ... with 1,139 more rows
```

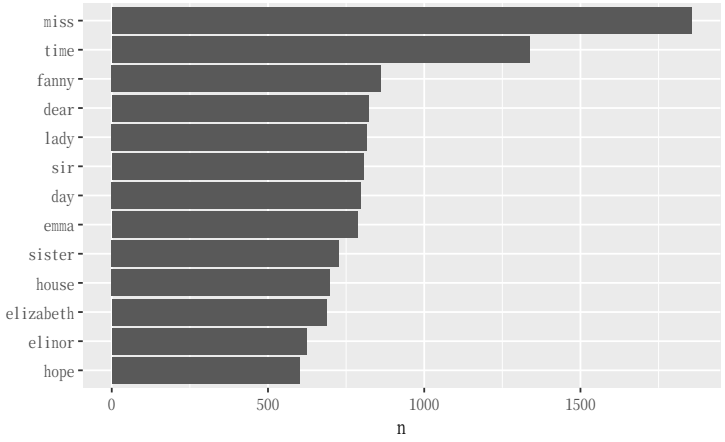
```
tidy_books <- tidy_books %>% anti_join(stop_words) # 巧用反连接删除
tidy_books
```

```
# A tibble: 217,609 x 4
   book                linenumber chapter word
   <fct>                    <int>   <int> <chr>
 1 Sense & Sensibility          1       0 sense
 2 Sense & Sensibility          1       0 sensibility
 3 Sense & Sensibility          3       0 jane
 4 Sense & Sensibility          3       0 austen
 5 Sense & Sensibility          5       0 1811
 6 Sense & Sensibility         10       1 chapter
 7 Sense & Sensibility         10       1 1
 8 Sense & Sensibility         13       1 family
 9 Sense & Sensibility         13       1 dashwood
10 Sense & Sensibility         13       1 settled
# ... with 217,599 more rows
```

```
## 统计词频
tidy_books %>% count(word, sort = TRUE)
```

```
# A tibble: 13,914 x 2
   word       n
   <chr>  <int>
 1 miss    1855
 2 time    1337
 3 fanny    862
 4 dear     822
 5 lady     817
 6 sir      806
 7 day      797
 8 emma     787
 9 sister   727
10 house    699
# ... with 13,904 more rows
```

```
## 绘图
tidy_books %>%
  count(word, sort = TRUE) %>% filter(n > 600) %>%
  mutate(word = reorder(word, n)) %>% # 按照n固定word的排序，因子化，
  ggplot(aes( word, n)) + geom_col() + xlab(NULL) + coord_flip()
```



1.3 情感分析

1.4 关键词

一篇文章中多次出现的高频词未必是文章的关键词，只有那些既比较高频又在其他语料中出现得相对少的词，才是我们要找的关键词。因此，衡量关键词需要在词频的基础上，对每个词分配一个重要性权重。最常见的词（如“的”、“在”、“了”、“我”）给予最小的权重，较常见的词（如“改革”、“创新”）给予较小的权重，较少见的词（如“非典”、“治理整顿”）给予较大的权重。

关键词的一个经典算法是 TF-IDF 算法，$TF-IDF = TF * IDF$。其中，TF (Term Frequency) 为经过标准化的词频，**TF= 某个词在文章中出现的次数/文章的总词数**或 **TF= 某个词在文章中出现的次数/出现次数最多的词的出现次数**。IDF (Inverse Document Frequency) 为逆文档频率，与该词出现在语料库中的次数负相关，一般令 **IDF = log(语料库文档总数/包含该词的文档数)**。这样，**TF-IDF 便与一个词在文档中的出现次数正相关，与该词在整个语料库的分布负相关**。对文档中的每个词计算 TF-IDF 的值，把结果从大到小排序，就得到了这篇文档的关键性排序列表。TF-IDF 值最大、排在最前面的几个词，就是这篇文章的关键词。

需要说明的是，tf-idf 是一个基于经验的统计量，缺乏令人信服的理论基础。

1.5 词之间的关系

1.5.1 n 元词组

1.5.2 相关词

1.6 文档–词项（document-term）矩阵

1.7 主题建模

2 中文分词技术

2.1 分词包 jiebaR

jiebaR Shiny APP: https://qinwf.shinyapps.io/jiebaR-shiny/

教程地址：http://qinwenfeng.com/jiebaR/

```
# 安装最新版
# library(devtools)
# install_github("qinwf/jiebaRD")
# install_github("qinwf/jiebaR")
# library("jiebaR")
```

结巴分词 (jiebaR)，是一款高效的 R 语言中文分词包，底层使用的是 C++，通过 Rcpp 进行调用很高效。

2.1.1 worker() 函数

worker(type = “mix”, dict = DICTPATH, hmm = HMMPATH, user = USERPATH, idf = IDFPATH, stop_word = STOPPATH, write = T, qmax = 20, topn = 5, encoding = “UTF-8”, detect = T, symbol = F, lines = 1e+05, output = NULL, bylines = F, user_weight = “max”)

- type, 引擎类型
- dict, 系统（分词）词典
- hmm, HMM 模型路径
- user, 用户词典
- idf, IDF 词典
- stop_word, 停止词词典
- write, 是否将文件分词结果写入文件，默认 FALSE
- qmax, 最大成词的字符数，默认 20 个字符
- topn, 关键词数, 默认 5 个
- encoding, 输入文件的编码，默认 UTF-8
- detect, 是否编码检查，默认 TRUE
- symbol, 是否保留符号，默认 FALSE
- lines, 每次读取文件的最大行数，用于控制读取文件的长度。大文件则会分次读取。
- output, 输出路径
- bylines, 按行输出
- user_weight, 用户权重

2.2 分词

2.2.1 分行输出

$bylines = T/F

```
# 新建一个分词器 wk
wk <- worker()
wk
```

```
Worker Type:  Jieba Segment

Default Method  :  mix
Detect Encoding :  TRUE
Default Encoding:  UTF-8
Keep Symbols    :  FALSE
Output Path     :
Write File      :  TRUE
By Lines        :  FALSE
Max Word Length :  20
Max Read Lines  :  1e+05

Fixed Model Components:

$dict
[1] "C:\\Users\\humoo\\AppData\\Local\\Temp\\Rtmpot2zpf/jiebaR_dict/dict/jieba.dict.utf

$user
[1] "C:/Users/humoo/Documents/R/win-library/3.6/jiebaRD/dict/user.dict.utf8"

$hmm
[1] "C:\\Users\\humoo\\AppData\\Local\\Temp\\Rtmpot2zpf/jiebaR_dict/dict/hmm_model.utf8

$stop_word
NULL
```

```
$user_weight
[1] "max"

$timestamp
[1] 1578446787

$default $detect $encoding $symbol $output $write $lines $bylines can be reset.
```

```
s1 <- segment(c("这是第一行文本。","这是第二行文本。"), wk)
s1
```

```
[1] "这是"   "第一行" "文本"   "这是"   "第二行" "文本"
```

```
# 分行输出，生成一个list
wk$bylines = TRUE
s2 <- segment(c("这是第一行文本。","这是第二行文本。"), wk)
s2
```

```
[[1]]
[1] "这是"   "第一行" "文本"

[[2]]
[1] "这是"   "第二行" "文本"
```

```
# 或
wk <- worker(bylines = TRUE)
s3 <- segment(c("这是第一行文本。","这是第二行文本。"), wk)
s3
```

```
[[1]]
[1] "这是"   "第一行" "文本"

[[2]]
[1] "这是"   "第二行" "文本"
```

2.2.2 保留标点符号

$symbol = T/F[1]

```
wk <- worker()
wk$symbol <- T
s1 <- segment(c("Hi, 这是第一行文本。"), wk)
s1
```

```
[1] "Hi"     ", "     "这是"   "第一行" "文本"   "。"
```

```
# 或
wk <- worker(symbol = T)
s2 <- segment(c("Hi, 这是第一行文本。"), wk)
s2
```

```
[1] "Hi"     ", "     "这是"   "第一行" "文本"   "。"
```

2.2.3 添加一个作为整体的新词，避免其被分开

2.2.3.1 方法一

new_user_word()

```
wk = worker()
segment("这是一个新词", wk)
```

```
[1] "这是" "一个" "新词"
```

```
new_user_word(wk, "这是一个新词", "n")
```

```
[1] TRUE
```

```
# 第三个参数 "n" 代表新词的词性标记
segment("这是一个新词", wk)
```

```
[1] "这是一个新词"
```

[1] 默认为 FALSE

2.2.3.2 方法二

自定义 user 词库，注意:

1. 编码格式必须为 utf-8
2. **必须空出首位两行，否则读进来的词会包含文件首位的格式符号**

```
words <- "想学R语言，那就赶紧拿起手机，打开微信，关注公众号《跟着菜鸟一起学R语言》，跟着
engine <- worker()
segment(words,engine)
```

```
 [1] "想学"   "R"      "语言"   "那"     "就"     "赶紧"   "拿起"
 [8] "手机"   "打开"   "微信"   "关注"   "公众号" "跟着"   "菜鸟"
[15] "一起"   "学"     "R"      "语言"   "跟着"   "菜鸟"   "一块"
[22] "飞"
```

```
readLines("dictionary.txt")
```

```
[1] "锘\xbf"      "R璇 "        "鍏 紬鍙\xb7" ""            ""
```

```
engine_user <- worker(user = 'dictionary.txt')
segment(words,engine_user)
```

```
 [1] "想学"   "R语言"  "那"     "就"     "赶紧"   "拿起"   "手机"
 [8] "打开"   "微信"   "关注"   "公众号" "跟着"   "菜鸟"   "一起"
[15] "学"     "R语言"  "跟着"   "菜鸟"   "一块"   "飞"
```

2.2.3.3 词性对照表

代码

名称

帮助记忆的诠释

Ag

形语素

形容词性语素。形容词代码为 a，语素代码 g 前面置以 A。

a

形容词

取英语形容词 adjective 的第 1 个字母。

ad

副形词

直接作状语的形容词。形容词代码 a 和副词代码 d 并在一起。

an

名形词

具有名词功能的形容词。形容词代码 a 和名词代码 n 并在一起。

b

区别词

取汉字 “别” 的声母。

c

连词

取英语连词 conjunction 的第 1 个字母。

Dg

副语素

副词性语素。副词代码为 d，语素代码 g 前面置以 D。

d

副词

取 adverb 的第 2 个字母，因其第 1 个字母已用于形容词。

e

叹词

取英语叹词 exclamation 的第 1 个字母。

f

方位词

取汉字 “方” 的声母。

g

语素

绝大多数语素都能作为合成词的 “词根”，取汉字 “根” 的声母。

h

前接成分

取英语 head 的第 1 个字母。

i

成语

取英语成语 idiom 的第 1 个字母。

j

简称略语

取汉字 “简” 的声母。

k

后接成分

l

习用语

习用语尚未成为成语，有点 “临时性”，取 “临” 的声母。

m

数词

取英语 numeral 的第 3 个字母，n，u 已有他用。

Ng

名语素

名词性语素。名词代码为 n，语素代码 g 前面置以 N。

n

名词

取英语名词 noun 的第 1 个字母。

nr

人名

名词代码 n 和 “人 (ren)” 的声母并在一起。

ns

地名

名词代码 n 和处所词代码 s 并在一起。

nt

机构团体

团的声母为 t，名词代码 n 和 t 并在一起。

nz

其他专名

专的声母的第 1 个字母为 z，名词代码 n 和 z 并在一起。

o

拟声词

取英语拟声词 onomatopoeia 的第 1 个字母。

p
介词
取英语介词 prepositional 的第 1 个字母。
q
量词
取英语 quantity 的第 1 个字母。
r
代词
取英语代词 pronoun 的第 2 个字母, 因 p 已用于介词。
s
处所词
取英语 space 的第 1 个字母。
Tg
时语素
时间词性语素。时间词代码为 t, 在语素的代码 g 前面置以 T。
t
时间词
取英语 time 的第 1 个字母。
u
助词
取英语助词 auxiliary 的第 2 个字母, 因 a 已用于形容词。
Vg
动语素
动词性语素。动词代码为 v。在语素的代码 g 前面置以 V。
v
动词
取英语动词 verb 的第一个字母。
vd
副动词
直接作状语的动词。动词和副词的代码并在一起。
vn
名动词
指具有名词功能的动词。动词和名词的代码并在一起。

w
标点符号
x
非语素字
非语素字只是一个符号，字母 x 通常用于代表未知数、符号。
y
语气词
取汉字“语”的声母。
z
状态词
取汉字“状”的声母的前一个字母。

2.2.4 添加停止词

停止词就是分词过程中，我们不需要作为结果的词，像英文的语句中有很多的 ‘a’, ‘the’, ‘or’, ‘and’ 等，中文语言中也有很多，比如“的”，“地”，“得”，“我”，“你”，“他”。这些词因为使用频率过高，会大量出现在一段文本中，对于分词后的结果，在统计词频的时候会增加很多的噪音，所以我们通常都会将这些词进行过滤。

在 jiebaR 中，过滤停止词有 2 种方法，一种是通过配置 stop_word 文件，另一种是使用 filter_segment() 函数。

2.2.4.1 方法一

目录下建立一个 stop.txt 文件，内容如下

```
# 配置stop_word文件
readLines("stop.txt")
```

```
[1] "锘\xbf" "鎴戣浠" "鐨\x84" "鍋滀"  "鍟\x8c" " "       ""        ""
```

```
wk <- worker()
segment("我说，这是一个停止词", wk)
```

```
[1] "我"    "说"    "这是" "一个" "停止" "词"
```

```
wk <- worker(stop_word = "stop.txt")
segment("我说，这是一个停止词", wk)
```

```
[1] "我"   "说"   "这是" "一个" "词"
```

2.2.4.2 方法二

```
# 动态调用filter_segment()函数
wk <- worker()
s2 <- segment("这是一个停止词", wk) %>%
  filter_segment('停止')
s2
```

```
[1] "这是" "一个" "词"
```

2.2.5 对文件分词

有两种方法，第一种借助于 readLines() 可以输出字符串或列表，再借助 writeBin(charToRaw()) 可以逐行保存，再输出为 txt；第二种只能输出 txt 文件。显然，第一种方法要更加灵活一些。

2.2.5.1 方法一

```
texts <- readLines("test.txt", encoding = "UTF-8")
wk <- worker(bylines = T) # 不可少
s1 = segment(texts, wk)
s1
```

```
[[1]]
character(0)

[[2]]
character(0)

[[3]]
```

```
[1] "这是" "一个" "停止" "词"

[[4]]
[1] "我"   "不"   "知道" "应该" "怎样"

[[5]]
character(0)
```

```
# 合并各行分词结果
col <- sapply(s1, function(x) {paste(x, collapse = " ")})
col
```

```
[1] ""                       ""                       "这是 一个 停止 词"
[4] "我 不 知道 应该 怎样" ""
```

```
# 再合并为一个包含换行符的大字符串
p <- paste(col, collapse = "\n")
p
```

```
[1] "\n\n这是 一个 停止 词\n我 不 知道 应该 怎样\n"
```

```
# 保存并查看
writeBin(charToRaw(p), "result1.txt")
readLines('result1.txt', encoding = "UTF-8")
```

```
[1] ""                       ""                       "这是 一个 停止 词"
[4] "我 不 知道 应该 怎样"
```

```
# 删除，防止程序反复运行，造成append式输出
file.remove("result1.txt")
```

```
[1] TRUE
```

2.2.5.2 方法二

```
## 第二种
wk$output = "result2.txt"
segment("test.txt", wk)
```

```
[1] "result2.txt"
```

```
readLines('result2.txt', encoding = "UTF-8")
```

```
[1] ""                      ""                      "这是 一个 停止 词"
[4] "我 不 知道 应该 怎样" ""
```

```
file.remove("result2.txt")
```

```
[1] TRUE
```

2.3 配置词典

对于分词的结果好坏的关键因素是词典，jiebaR 默认有配置标准的词典。对于我们的使用来说，不同行业或不同的文字类型，最好用专门的分词词典。在 jiebaR 中通过 show_dictpath() 函数可以查看默认的标准词典。

```
# 查看默认的词库位置
show_dictpath()
```

```
[1] "C:/Users/humoo/Documents/R/win-library/3.6/jiebaRD/dict"
```

```
# 查看目录中所有文件
dir(show_dictpath())
```

```
[1] "C:/Users/humoo/Documents/R/win-library/3.6/jiebaRD/dict"
```

```
[1] "backup.rda"      "hmm_model.zip"   "idf.zip"         "jieba.dict.zip"
[5] "model.rda"       "README.md"       "stop_words.utf8" "user.dict.utf8"
```

词典目录中，包括了多个文件。

- jieba.dict.utf8, 系统词典文件，最大概率法，utf8 编码的

- hmm_model.utf8, 系统词典文件，隐式马尔科夫模型，utf8 编码的
- user.dict.utf8, 用户词典文件，utf8 编码的
- stop_words.utf8，停止词文件，utf8 编码的
- idf.utf8，IDF 语料库，utf8 编码的
- jieba.dict.zip，jieba.dict.utf8 的压缩包
- hmm_model.zip，hmm_model.utf8 的压缩包
- idf.zip，idf.utf8 的压缩包
- backup.rda，无注释
- model.rda，无注释
- README.md，说明文件

2.4 词频统计与词云绘制

2.4.1 词频统计

jiebaR::freq()

输入字符串向量，返回词频统计数据框

```
word_freq <- c(rep("中国", 10), rep("宏观",8), rep("经济", 5),
               rep('贸易', 4), rep("AI", 3), rep('IT', 5),
               rep("ICT", 2), rep('5G', 7))
df <- jiebaR::freq(word_freq)
df
```

```
  char freq
1   5G    7
2  ICT    2
3   AI    3
4 贸易    4
5 宏观    8
6   IT    5
7 经济    5
8 中国   10
```

2.4.2 绘制词云

2.4.2.1 wordcloud2 包

官方介绍见https://www.r-graph-gallery.com/196-the-wordcloud2-library

wordcloud2(data, size = 1, minSize = 0, gridSize = 0, fontFamily = 'Segoe UI', fontWeight = 'bold', color = 'random-dark', backgroundColor = "white", minRotation = -pi/4, maxRotation = pi/4, shuffle = TRUE, rotateRatio = 0.4, shape = 'circle', ellipticity = 0.65, widgetsize = NULL, figPath = NULL, hoverFunction = NULL)

- shape: 可以选择词云的形状，有上面代码可知它默认为圆形（circle），它还提供了其他一些参数，cardioid(心形)，star(星形)，diamond（钻石形），triangle-forward（三角形），triangle（三角形），这两个三角形就是倾斜方向不同而已，pentagon(五边形)。
- figPath: 一张黑白图，词云自动填充在黑色区域。figPath 为黑白图的路径

```
wordcloud2(df, size = 0.6, fontFamily = "SimHei",
           color = "random-light", backgroundColor = "grey")
```

\begin{center}

```
wordcloud2(demoFreqC, size = 1, fontFamily = "微软雅黑",
           color = "random-light", backgroundColor = "grey")
```

2.4.2.2 wordcloud2 词云对象的保存

wordcloud2() 生成的图像是用 js 渲染的，其类型为 html widget，因此不能用一般的方法保存。只能（1）先将其保存为一个 html 文件，（2）再用对 html 截图的方式保存。这两个过程分别要用到 `htmlwidgets::saveWidget()` 和 `webshot::webshot()`。

```
library(webshot)
# webshot::install_phantomjs()
# 第一次使用webshot包还需要安装额外的东西

# 生成 wordcloud2 图像，my_graph 的类型是 htmlwidget
my_graph <- wordcloud2(demoFreq, size = 1.5)

# htmlwidgets::saveWidget() 可以将 widget 保存为一个 HTML 文件
library("htmlwidgets")
htmlwidgets::saveWidget(my_graph,"tmp.html",selfcontained = F)
```

注意，参数 selfcontained 必须为 F，才能保存一些 external resources, 体现为与 tmp.html 同级的 tmp_files 文件夹中的.css 和.js 文件。有了它们，才能正确完成页面的渲染

```
# webshot::webshot() 用于对一个网页截图
# delay 参数即等待的秒数，等待几秒后再截图，因为可能页面渲染需要时间
webshot("tmp.html", "fig_1.png", delay = 1,
        vwidth = 1000, vheight = 750)
```

\begin{center}

```
webshot("tmp.html", "fig_2.png", delay = 2,
        vwidth = 992, vheight = 744)
```


```
webshot("tmp.html", "fig_5.png", delay = 5,
        vwidth = 1000, vheight = 750)
```


```
webshot("tmp.html", "fig_10.png", delay = 10,
        vwidth = 1000, vheight = 750)
```


```
# 可以清楚地看到，等待秒数越多，词云图越精细。
```

2.5 关键词提取

若不自己写函数定义 IDF 值，则可以使用默认的 IDF 表，储存在 jiebaR 包文件夹中。

方法一：

worker() 定义提取器，keywords() 提取

```
key <- worker("keywords", user = 'dictionary.txt',
              topn = 10) # topn 即取前几个关键词
keywords("我深入钻研了半年R语言", key)
```

```
11.7392 9.20933 6.74841 6.39904
"R语言"  "钻研"  "半年"  "深入"
```

由于字符串中三个词都是只出现了一次，所以 TF-IDF 的值反映了 IDF 值的大小。

方法二：

segment() %>% vector_keywords() 提取

```
wk <- worker()
s <- segment("我深入钻研了半年R语言", wk)
s
```

```
[1] "我"    "深入" "钻研" "了"    "半年" "R"     "语言"
```

```
key <- worker("keywords", topn = 10)
vector_keywords(s, key)
```

```
11.7392 9.20933 6.74841 6.39904  6.1635
    "R"  "钻研"  "半年"  "深入"  "语言"
```

```
# vector_keywords()将分词结果和关键词提取引擎结合起来
```

可见，**提取关键词时，与单纯分词不同，会自动删除停止词**

3 综合案例：1954-2019 年政府工作报告

历年国务院政府工作报告合集网址：http://www.gov.cn/guowuyuan/baogao.htm

3.1 爬虫抓取

```
url <- 'http://www.gov.cn/guowuyuan/baogao.htm'
web <- read_html(url, encoding = "utf-8")

nodes <- web %>%
  html_nodes("#UCAP-CONTENT a") %>%
  html_attrs() %>% purrr::map_chr(1) %>%
  str_sub(1L, -3L) # 各篇报告的网址
n <- length(nodes) # 统计一下共有多少篇


## 自定义抓取函数
```

```
# 准备，标题级别对应表
level1 <- data.table(start = c('一、', '二、', '三、', '四、', '五、', '六、', '七、',
level2 <- data.table(start = c('（一', '（二', '（三', '（四', '（五', '（六', '（七',
level <- rbind(level1, level2)

# 准备：年份与总理姓名对应表
premier <- tibble(year = 1954:2019, author = '') %>%
  filter(year < 1961 | year == 1964 | year == 1975 | year > 1977) %>%
  setDT()
premier[year < 1976, author := '周恩来']
premier[year == 1955, author := '李富春']
premier[year == 1956, author := '李先念']
premier[year == 1958, author := '薄一波']
premier[year == 1960, author := '谭振林']
premier[year %in% c(1978, 1979), author := '华国锋']
premier[year == 1980, author := '姚依林']
premier[year %in% 1981:1987, author := '赵紫阳']
premier[year %in% 1988:1998, author := '李鹏']
premier[year %in% 1999:2003, author := '朱镕基']
premier[year %in% 2004:2013, author := '温家宝']
premier[year %in% 2014:2019, author := '李克强']

# 抓取文献函数，返回一个数据框
crawl <- function(url){
  text <- url %>% read_html(encoding = "utf-8") %>%
    html_nodes("p") %>% html_text() %>%
    str_trim() %>% str_remove_all(' ') %>% # 去掉各种空格
    str_replace_all(c("１" = "1", "２" = "2", "３" = "3",
                      '４' = '4', '５' = '5', '６' = '6',
                      '７' = '7', '８' = '8', '９' = '9',
                      '０' = '0')) # 换为半角数字
  year <- text %>% str_c(collapse = '') %>%
```

```
    str_extract("\\d{4}年\\d{1,2}月\\d{1,2}日") %>%
    str_sub(1, 4) %>% as.numeric()
  df <- tibble(text = text) %>% mutate(year = year) %>%
    filter(text != '') %>%
    mutate(start = str_sub(text, 1, 2)) %>%
    left_join(level, by = 'start') %>%
    mutate(level = ifelse(is.na(level), 0, level)) %>%
    mutate(title = str_c(year, '年国务院政府工作报告')) %>%
    left_join(premier, by = 'year') %>%
    select(year, title, author, level, text)
  return(df)
}

## 抓取所有节点背后的政府工作报告，构成一个list
gov_report <- map(nodes, crawl)
```

3.2 保存进文献数据库

```
# 保存1959年以来的（之前的不够规范）国务院政府工作报告
channel <- dbConnect(SQLite(),
  dbname = "C:/Users/humoo/OneDrive/ICT/DataBase/text.db")

dbSendQuery(channel, 'DROP TABLE IF EXISTS gov_report')

# 建表
dbSendQuery(conn = channel,
"CREATE TABLE gov_report
(year INTEGER,
title TEXT,
author TEXT,
level INTEGER,
text TEXT)")
```

```
# 写入
save2db <- function(db){
  dbWriteTable(channel, "gov_report", db, append = T)
}
map(gov_report, save2db)

# 去掉1957年之前的
dbSendQuery(channel, "DELETE FROM gov_report WHERE year < 1957;")

dbDisconnect(channel) # 断开连接
rm(list = ls())
```

3.3 高频词和关键词分析

3.3.1 高频词

```
channel <- dbConnect(SQLite(),
  dbname = "C:/Users/humoo/OneDrive/ICT/DataBase/text.db")
report <- dbReadTable(channel, "gov_report") %>% setDT()

# 报告年份的遍历向量
traversal <- sqldf("SELECT DISTINCT year FROM report")[,1] %>%
  as.vector()

# 分词引擎
engine <- worker(user = 'user.txt',
                 stop_word = "stop_words.utf8")

# 自定义词频提取函数
get_tf <- function(year_input) {
  # 1 组合一篇报告为一个长字符串
  string <- report[year == year_input]$text %>%
    str_c(collapse = '') %>%
```

```
    str_replace_all('\\d',"") # 去掉所有的数字
  # 2 对长字符串分词
  jieba <- segment(string, engine) %>% freq() %>% setDT()
  # 3 计算经过标准化的tf值
  tf <- jieba[,list(year = year_input, words = char, tf = freq/sum(freq))]
  # 4 排序并返回
  tf[order(tf, decreasing = T)] %>% return()
}

# 各年份词频列表
list_tf <- map(traversal, get_tf)
names(list_tf) <- traversal %>% as.character()

# 每年政府工作报告的十大高频词
top10_tf <- function(df){
  df %>% head(10) %>% return()
}
list_tf10 <- map(list_tf, top10_tf)
names(list_tf10) <- traversal %>% as.character()
```

3.3.2 关键词

注：idf 的计算公式是一个经验公式，没有多少理论基础。

$idf = lg(48/count))$，刚好可以使得在所有文档中都出现过的词的 idf 为零，从而保证了不会被排入关键词。

```
## 合并48篇报告的词库，并去掉了4千多个单字
total_words <- reduce(list_tf,rbind)[str_length(words) > 1]
## 求总词库，配以统计量：每个词在48篇报告中的多少篇中出现过
golssary <- total_words %>%
  count(words, sort = T, name = 'count') %>%
  mutate(idf = log(48/count)) # 计算一个词的IDF
# 也可以用sql语句：golssary <- sqldf("SELECT words, COUNT() AS count FROM total_words G
```

```
# 每年政府工作报告的三十大关键词
get_idf <- function(year_input){
  list_tf[[year_input %>% as.character()]] %>%
    left_join(golssary) %>% mutate(tf_idf = tf*idf) %>%
    arrange(desc(tf_idf)) %>% select(year, words, tf_idf) %>%
    head(30)
}
list_idf30 <- map(traversal, get_idf)
names(list_idf30) <- traversal %>% as.character()
```

```
## 绘制词云图
for (i in traversal) {
  list_idf30[[i %>% as.character()]] %>%
    select(-year) %>%
    wordcloud2(size = 0.8, fontFamily = "SimHei",
               color = "random-light",
               backgroundColor = "grey") %>%
    saveWidget("tmp.html",selfcontained = F)
  path <- str_c('Figures/tf-idf_', i, '.png')
  webshot("tmp.html", path, delay = 3,
          vwidth = 1000, vheight = 750)
}

save(list_tf10, list_idf30, traversal, file = 'C:/Users/humoo/OneDrive/ICT/shinyapp/Gov
```

```
## 将list_tf10和list_idf30数据保存为csv文件，用以进行动态可视化
csv_idf <- reduce(list_idf30, rbind) %>% mutate(type = 'TF-IDF') %>%
  select(words, type, tf_idf, year) %>%
  rename(name = words, value = tf_idf, date = year) %>%
  mutate(value = value * 1000, date = as.Date(str_c(date, '-01-01')))

# 用write_excel_csv()，默认UTF-8编码，避免乱码
```

```
write_excel_csv(csv_idf, 'C:/Users/humoo/OneDrive/ICT/shinyapp/Government_Report/data/k


csv_tf <- reduce(list_tf10, rbind) %>% mutate(type = 'TF') %>%
  select(words, type, tf, year) %>%
  rename(name = words, value = tf, date = year) %>%
  mutate(value = value * 100, date = as.Date(str_c(date, '-01-01')))

write_excel_csv(csv_tf, 'C:/Users/humoo/OneDrive/ICT/shinyapp/Government_Report/data/tf
```