# 字符串操作：正则表达式及 stringr 程序包

*zmt*

*2017 年 3 月 1 日*

## 0. 为什么学习字符串处理

文本分析与文本挖掘是数据挖掘领域的一个重要分支。我们日常生活中接触到的大部分数据都是以文本形式存在。文本分析与文本挖掘在业界中也有着非常广泛的应用。然而，传统的统计学教育几乎没有告诉过我们，如何进行文本的统计建模分析。

由于文本数据属于非结构化数据，要想对文本数据进行传统的统计分析，必须经过层层的数据清洗与整理。

今天我们要介绍的字符串操作就是用来干这一种脏活累活的。

与建立酷炫的模型比起来，数据的清洗与整理似乎是一种低档次的工作。如果把数据分析师比作厨师，那么，数据清洗无疑是类似洗菜切菜的活儿。然而想要成为资深的『数据玩家』，这种看似低档次的工作是必不可少的，并且，这种工作极有可能占据你整个建模流程 80% 的时间。

如果能够掌握高效的数据清洗工具，我们将拥有更多的时间来进行模型选择和参数调整，使得我们的模型更加合理有效。

此外，对于不需要进行文本建模分析的同学，掌握文本处理的工具也将对减轻你的工作负担大有益处。下面是几个文本处理工具让生活更美好的例子：

- R 辅助邮件查阅及核对：R 字符串处理应用之邮件考勤自动化
- R 辅助 SAS 处理大量数据：深入理解 SAS 之批量数据导入
- R 爬虫获取数据：这个将在后续的课程中作为一个专题进行介绍。

可见，我们可以用到文本处理工具的场景还是非常多的，如批量文件名修改、批量字符替换、大量邮件或 html 文件处理等。

下面，我将通过一个例子，展示 R 字符串处理的大致功能。接着，介绍正则表达式的基础概念。然后，介绍 R 字符串处理中一个非常好用的程序包 `stringr`，并接着介绍一些与文件编码处理相关的函数。最后，通过一个案例展示字符串处理的真实应用场景。

## 1. 初步认识 R 中的字符串处理

为了先给大家一个关于字符串处理的大体认识，我们使用 R 自带的一个数据集 `USArrests` 进行函数功能演示。

先看看数据集的结构。

```
# take a glimpse
head(USArrests)
```

字符串子集提取：获得州的简称

```
# 获得州名
states = rownames(USArrests)

# 方法一：substr()
substr(x = states, start = 1, stop = 4)

# 方法二：abbreviate()

abbreviate(states,minlength = 5)
```

字符统计：获得名字最长的州名

```
# 获得每个州名的长度

state_chars <- nchar(states)

# 绘制直方图
hist(state_chars, main = "Histogram",
     xlab = "number of characters in US State names")

# 最长的州名
states[which(state_chars == max(state_chars))]
```

**注意：nchar() 与 length() 的区别**

字符串匹配：含某些字母的州名

```
# 获得名字里含有'w' 的州
grep(pattern = "w", x = states, value = TRUE)

############################

# 获得名字里含有'W' 或'w' 的州

## 方法一：
grep(pattern = "[wW]", x = states, value = TRUE)

## 方法二：
grep(pattern = "w", x = tolower(states),value = TRUE)

## 方法三：
grep(pattern = "W", x = toupper(states), value = TRUE)
```

```
## 方法四：
grep(pattern = "w",x = states, ignore.case = TRUE, value = TRUE)
```

字符统计：某些字母个数统计

```
library(stringr)

# 每个州名里含字母'a' 的数目
str_count(states,"a")

##################
# 计算元音字母的数目

# 新建元音向量
vowels <- c("a","e","i","o","u")

# 新建一个空向量用来存储结果
num_vowels <- vector(mode = "integer",length = 5)

# 计算每个元音的数目
for(i in seq_along(vowels)){
  num_aux <- str_count(tolower(states),vowels[i])
  num_vowels[i] <- sum(num_aux)
}

# 给向量添加名称
names(num_vowels) <- vowels

# 元音字母数目统计
num_vowels

# 绘制条形图
barplot(num_vowels, main = "number of vowels in USA States names")
```

## 2. 正则表达式

正则表达式是对字符串类型数据进行匹配判断、提取等操作的一套逻辑公式。

处理字符串类型数据方面，高效的工具有 Perl 和 Python。如果我们只是偶尔接触文本处理任务，学习 Perl 无疑成本太高；如果常用 Python，则可以利用成熟的正则表达式模块：re库；如果常用 R，则使用 Hadley 大神开发的 `stringr` 包就已经游刃有余。

下面，我们先简要介绍重要且通用的正则表达式规则。接着，总结一下 `stringr`包中需要输入正则表

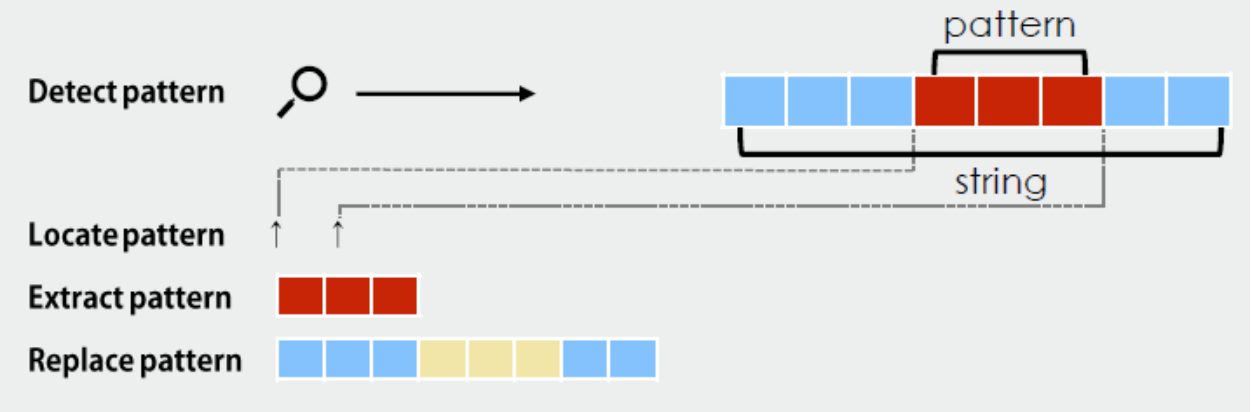


Figure 1: 正则表达式原理图

达式参数的字符处理函数。

### 元字符 (Metacharacters)

大部分的字母和所有的数字都是匹配它们自身的正则表达式。然而，在正则表达式的语法规定中，有 12 个字符被保留用作特殊用途。它们分别是：

```
[ ] \ ^ $ . | ? * + ( )
```

如果我们直接进行对这些特殊字符进行匹配，是不能匹配成功的。正确理解它们的作用与用法，至关重要。

**以下用法都是错误的!!**

```
metaChar = c("$","*","+",".","?","[","^","{","|","(","\\")

grep(pattern = "$", x = metaChar, value = TRUE)

grep(pattern = "\\", x = metaChar, value = TRUE)

grep(pattern = "(", x = metaChar, value = TRUE)

gsub(pattern = "|", replacement = ".", "gsub|uses|regular|expressions")

strsplit(x = "strsplit.also.uses.regular.expressions", split = ".")
```

它们的作用如下：

- `[ ]`：括号内的任意字符将被匹配；

```
# example
grep(pattern = "[wW]", x = states, value = TRUE)
```

- `\`：具有两个作用：

  - 1. 对元字符进行转义 (后续会有介绍)
  - 2. 一些以 \ 开头的特殊序列表达了一些字符串组

```
strsplit(x = "strsplit.also.uses.regular.expressions", split = ".")

# compare
strsplit(x = "strsplit.also.uses.regular.expressions", split = "\\.")

################
# function 2:

str_extract_all(string = "my credit card number: 34901358932236",pattern = "\\d")
```

- ^：匹配字符串的开头，将 ^ 置于 character class 的首位表达的意思是取反义。如 [^5] 表示匹配除了 “5” 以外的所有字符。

```
# function 1
test_vector <- c("123","456","321")

str_extract_all(test_vector,"3")
str_extract_all(test_vector,"^3")

# function 2
str_extract_all(test_vector,"[^3]")
```

- $：匹配字符串的结尾。但将它置于 character class 内则消除了它的特殊含义。如 [akm$]将匹配'a','k','m' 或者'$'。

```
# function 1
test_vector <- c("123","456","321")

str_extract_all(test_vector,"3$")



# function 2
str_extract_all(test_vector,"[3$]")
```

- .：匹配除换行符以外的任意字符。

```
str_extract_all(string = c("regular.expressions\n","\n"), pattern =".")
```

- |：或者

```
test_vector2 <- c("WiseRclub 很厉害！ ", "wiserclub 是啥",
        " 现在叫 WiseR club, 不只是 R 语言的 club。")
str_extract_all(string = test_vector2, pattern ="WiseRclub|WiseR club|R 语言")
```

- ?：此符号前的字符 (组) 是可有可无的，并且最多被匹配一次

```
str_extract_all(string = c("abc","ac","bc"),pattern = "ab?c")
```

- ( )：表示一个字符组，括号内的字符串将作为一个整体被匹配。

```
str_extract_all(string = c("abc","ac","cde"),pattern = "(ab)?c")

str_extract_all(string = c("abc","ac","cde"),pattern = "ab?c")
```

- *：此符号前的字符 (组) 将被匹配零次或多次

```
str_extract_all(string = c("abababab","abc","ac"),pattern = "(ab)*")
```

- +：前面的字符 (组) 将被匹配一次或多次

```
str_extract_all(string = c("abababab","abc","ac"),pattern = "(ab)+")
```

### 用于匹配重复字符串的符号

| 代码 | 含义说明 |
|---|---|
| ? | 重复零次或一次 |
| * | 重复零次或多次 |
| + | 重复一次或多次 |
| {n} | 重复 n 次 |
| {n,} | 重复 n 次或更多次 |
| {n,m} | 重复 n 次到 m 次 |

```
str_extract_all(string = c("abababab","ababc","abc"),pattern = "(ab){2,3}")
```

### 转义

如果我们想查找元字符本身，如 “?” 和 “*“，我们需要提前告诉编译系统，取消这些字符的特殊含义。这个时候，就需要用到转义字符 \，即使用 \?和 *。当然，如果我们要找的是 \, 则使用 \\ 进行匹配。

```
strsplit(x="strsplit.also.uses.regular.expressions", split=".")

# compare
strsplit(x="strsplit.also.uses.regular.expressions", split="\\.")
```

**注：R 中的转义字符则是双斜杠：**

### R 中预定义的字符组

| 代码 | 含义说明 |
|---|---|
| [:digit:]或 \\d | 数字; [0-9] |
| [^[:digit:]]或 \\D | 非数字; 等价于 [^0-9] |
| [:lower:] | 小写字母; [a-z] |
| [:upper:] | 大写字母; [A-Z] |
| [:alpha:] | 字母; [a-z]及 [A-Z] |
| [:alnum:] | 所有字母及数字; [A-z0-9] |
| \\w | 字符串; [A-z0-9_] (在 ASCII 编码下,\w比 [:alnum:]多了一个下划线) |
| [:xdigit:]或 \\x | 十六进制数字; [0-9A-Fa-f] |
| [:punct:] | 标点符号; ! " ' # $ % & ^ () * +等 |
| [:graph:] | Graphical characters, 即 [:alnum:]和 [:punct:] |
| [:blank:] | 空字符; 即 Space 和 Tab |
| [:space:]或 \\s | Space，Tab，newline，及其他 space characters |
| [:print:] | 可打印的字符，即：[:alnum:]，[:punct:]和 [:space:] |
| ——- | —— |
| \\b | Empty string at either edge of a word（单词开头或结束的位置） |
| \\B | Not the edge of a word（非单词开头或结束的位置） |
| \\< | Beginning of a Word（单词开头的位置） |
| \\> | End of a word（单词结束的位置） |

```
str_extract_all(string = "my credit card number: 34901358932236",pattern = "[:digit:]")
```

**特殊用途的字符**

| 代码 | 含义说明 |
|---|---|
| \n | 换行 |
| \r | 回车 |
| \t | 缩进 |
| \v | 垂直缩进 |
| \f | 换页 |
| \b | 退格 |

Tips：回车与换行的区别

## 3.stringr字符串处理函数对比学习

**stringr包中的重要函数**

| 函数 | 功能说明 | R Base 中对应函数 |
|---|---|---|
| 使用正则表达式的函数 | | |
| str_extract() | 提取首个匹配模式的字符 | regmatches() |
| str_extract_all() | 提取所有匹配模式的字符 | regmatches() |
| str_locate() | 返回首个匹配模式的字符的位置 | regexpr() |
| str_locate_all() | 返回所有匹配模式的字符的位置 | gregexpr() |
| str_replace() | 替换首个匹配模式 | sub() |
| str_replace_all() | 替换所有匹配模式 | gsub() |
| str_split() | 按照模式分割字符串 | strsplit() |
| str_split_fixed() | 按照模式将字符串分割成指定个数 | - |
| str_detect() | 检测字符是否存在某些指定模式 | grepl() |
| str_count() | 返回指定模式出现的次数 | - |
| 其他重要函数 | | |
| str_sub() | 提取指定位置的字符 | regmatches() |
| str_dup() | 丢弃指定位置的字符 | - |
| str_length() | 返回字符的长度 | nchar() |
| str_pad() | 填补字符 | - |
| str_trim() | 丢弃填充，如去掉字符前后的空格 | - |
| str_c() | 连接字符 | paste(),paste0() |

可见，stringr包中的字符处理函数更丰富和完整，并且更容易记忆。

### 文本文件的读写

这里的文本文件指的是非表格式的文件，如纯文本文件、html 文件。文本文件的读取可以使用readLines()和 scan()函数。一般需要通过 encoding参数设置文件内容的编码方式。

```
# 假设当前路径有一个文件为 `file.txt`
text <- readLines("file.txt", encoding = "UTF-8")

# 默认设置，每个单词作为字符向量的一个元素
scan("file.txt", what = character(0),encoding = "UTF-8")

# 设置成每一行文本作为向量的一个元素，这类似于 readLines
scan("file.txt", what = character(0), sep = "\n", encoding = "UTF-8")

# 设置成每一句文本作为向量的一个元素
scan("file.txt", what = character(0), sep = ".", encoding = "UTF-8")
```

文本文件的写出可以使用 cat()和 writeLines()函数。

```r
# 假设要保存当前环境中的 R 变量 text
# sep 参数指定要保存向量里的元素的分割符号。
cat(text, file = "file.txt", sep = "\n")

writeLines(text, con = "file.txt", sep = "\n", useBytes = F)
```

字符统计及字符翻译

```r
x <- c("I love R","I'm fascinated by Statisitcs")

###################
## 字符统计

# nchar
nchar(x)

# str_count
str_count(x)
str_length(x)
#######################

DNA <- "AgCTaaGGGcctTagct"

## 字符翻译：大小写转换
tolower(DNA)
toupper(DNA)

## 字符翻译：符号替换（逐个替换）
# chartr
chartr("Tt", "Uu", DNA)  # 将 T 碱基替换成 U 碱基

# 注意：与 str_replace() 的区别
str_replace_all(toupper(DNA), pattern = "T", replacement = "U")
```

字符串连接

```r
# paste
paste("control", 1:3, sep = "_")

# str_c()
```

```r
str_c("control", 1:3, sep = "_")
```

### 字符串查询

字符串的查询或者搜索应用了正则表达式的匹配来完成任务。R Base 包含的字符串查询相关的函数有 grep()，grepl()，regexpr()，gregexpr()和 regexec()等。

```r
##################################
## 包含匹配

# grep
x <- c("I love R", "I'm fascinated by Statisitcs", "I")
grep(pattern = "love", x = x)
grep(pattern = "love", x = x, value = TRUE)
grepl(pattern = "love", x = x)

# str_detect
str_detect(string = x, pattern = "love")

##################################
#
# match, 完全匹配， 常用的 %in% 由 match() 定义
match(x = "I",table = x)
"I" %in% x
```

### 字符串拆分

```r
# strsplit
text <- "I love R.\nI'm fascinated by Statisitcs."
cat(text)
strsplit(text,split = " ")
strsplit(text,split = "\\s")

# str_split
str_split(text,pattern = "\\s")
```

### 字符串替换

sub() 和 gsub() 能够提供匹配替换的功能，但其替换的实质是先创建一个对象，然后对原始对象进行重新赋值，最后结果好像是“替换”了一样。

sub() 和 gsub() 的区别在于，前者只替换第一次匹配的字串，而后者会替换掉所有匹配的字串。

也可以使用 substr 和 substring 对指定位置进行替换。

```
####################################
## 匹配替换

test_vector3 <- c("Without the vowels,We can still read the word.")


# sub
sub(pattern = "[aeiou]",replacement = "-",x = test_vector3)

# gsub
gsub(pattern = "[aeiou]",replacement = "-",x = test_vector3)

# str_replace_all
str_replace_all(string = test_vector3,pattern = "[aeiou]",
                replacement = "")
```

### 字符串提取

常用到的提取函数有 substr() 和 substring()，它们都是靠位置来进行提取的，它们自身并不适用正则表达式，但是它们可以结合正则表达式函数 regexpr()，gregexpr() 和 regexec() 等可以方便地从文本中提取所需信息。

stringr 包中的函数 str_sub 和 str_dup 可以通过位置提取，而 str_extract 和 str_match 可以通过正则表达式提取。

```
########################################
## 指定位置替换
substr("abcdef", start = 2, stop = 4)
substring("abcdef", first = 1:6, last = 1:6)

str_sub("abcdef",start = 2, end = 4)
str_sub("abcdef",start = 1:6, end = 1:6)

################################

text_weibo <- c("# 围棋人机大战 # 【人工智能攻克围棋 AlphaGo 三比零完胜李世石】",
                " 谷歌人工智能 AlphaGo 与韩国棋手李世石今日进行了第三场较量",
                " 最终 AlphaGo 战胜李世石，连续取得三场胜利。接下来两场将沦为李世石的“荣誉之战”。")

# str_match_all, 返回的列表中的元素为矩阵
str_match_all(text_weibo, pattern = "#.+#")
```

```r
str_match_all(text_weibo, pattern = "[a-zA-Z]+")

# str_extract_all, 返回的列表中的成分为向量
str_extract_all(text_weibo, pattern = "#.+#")

str_extract_all(text_weibo, pattern = "[a-zA-Z]+")
```

#### 字符串定制输出

这个内容有点类似于字符串的连接。R 中相应的函数为 strtrim()，用于将字符串修剪到特定的显示宽度。stringr 中相应的函数为：str_pad()。

strtrim() 会根据 width 参数提供的数字来修剪字符串，若 width 提供的数字大于字符串的字符数的话，则该字符串会保持原样，不会增加空格之类的东西, 若小于，则删除部分字符。而 str_pad() 则相反。

```r
strtrim(c("abcde", "abcde", "abcde"),width =  c(1, 5, 10))

str_pad(string = c("abcde", "abcde", "abcde"),width =  c(1, 5, 10),side = "right")
```

strwrap() 会把字符串当成一个段落来处理（不管段落中是否有换行），按照段落的格式进行缩进和分行，返回结果就是一行行的字符串。

而 str_wrap() 没有直接把文本切割成向量，而是在文本内容中插入了缩进或分行的标识符。

```r
string <- "Each character string in the input is first split into\n paragraphs (or lines containing whi

strwrap(x = string, width = 30)

# str_wrap
str_wrap(string = string, width = 30)
cat(str_wrap(string = string, width = 30))
```

## 4. 字符编码相关的重要函数

windows 下处理字符串类型数据最头疼的无疑是编码问题了。这里介绍几个编码转换相关的函数。

| 函数 | 功能说明 |
|---|---|
| iconv() | 转换编码格式 |
| Encoding() | 查看或者指定编码格式 |
| tau::is.locale() | 检测字符向量元素是否在当前区域的编码中 |
| tau::is.utf8() | 检测字符向量元素是否为 UTF-8 编码的字符串 |

虽然查看编码方式已经有 Encoding()函数，但是这个函数往往在很多时候都不灵，经常返回恼人的

“Unknow”。而火狐浏览器进行网页文本编码识别的一个 c++ 库 universalchardet ，可以识别的编码种类较多。厦大 2011 级的覃文锋学长写过一个相应的 R 包接口，专用于文件编码方式检测，具体请参考：checkenc - 自动文本编码识别

```
devtools::install_github("qinwf/checkenc")
library(checkenc)
checkenc("2016-03-10-regular-expression-and-strings-processing-in-R.html")
```

## 5. 应用案例

最后，给大家展示一个小小的爬虫案例：爬取豆瓣 T250 中的电影信息进行分析。这里出于练习的目的刻意使用了字符串处理函数，在实际的爬虫中，有更方便快捷的实现方式。

本案例改编自肖凯老师的博客在 R 语言中使用正则表达式，原博客使用 R Base中的函数进行处理字符串，这里已经全部更改为 stringr中的函数进行处理。

```
url <-'http://movie.douban.com/top250?format=text'
# 获取网页原代码，以行的形式存放在 web 变量中
web <- readLines(url,encoding="UTF-8")
# 找到包含电影名称的行
name <- str_extract_all(string = web, pattern = '<span class="title">.+</span>')
movie.names_line <- unlist(name)
# 用正则表达式来提取电影名
movie.names <- str_extract(string = movie.names_line, pattern = ">[^&].+<") %>%
        str_replace_all(string = ., pattern = ">|<",replacement = "")

movie.names <- na.omit(movie.names)

# 获取评价人数
Rating <- str_extract_all(string = web,pattern = '<span>[:digit:]+ 人评价</span>')

Rating.num_line <- unlist(Rating)

Rating.num <- str_extract(string = Rating.num_line, pattern = "[:digit:]+") %>% as.numeric(.)

# 获取评价分数
Score_line <- str_extract_all(string = web,
                              pattern = '<span class="rating_num" property="v:average">[\\d\\.]+</span>')

Score_line <- unlist(Score_line)

Score <- str_extract(string = Score_line, pattern = '\\d\\.\\d') %>%
  as.numeric(.)
```

```
# 数据合并

MovieData <- data.frame(MovieName = movie.names,
                        RatingNum = Rating.num,
                        Score = Score,
                        Rank = seq(1,25),stringsAsFactors = FALSE)

# 可视化
library(ggplot2)
ggplot(data = MovieData, aes(x = Rank,y = Score)) +
        geom_point(aes(size = RatingNum)) + geom_text(aes(label = MovieName),
        colour = "blue", size = 4, vjust = -0.6)
```

## 深入学习

- R Wikibook: Programming and Text Processing
- Introduction to stringr
- 正则表达式 30 分钟入门教程
- 深入浅出之正则表达式

## 参考文献


- Handling and Processing Strings in R
- Automated Data Collection in R
- R 中的普通文本处理 -汇总
- checkenc - 自动文本编码识别
- 在 R 语言中使用正则表达式