正则表达式及R字符串处理

yphuang

*2016*年*3*月*10*日

0.动机：为什么学习字符串处理

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。然而，我们日常生活中接触到的大部分数据都是以文本的形式存在。文本分析与挖掘在业界中也有着非常广泛的应用。

由于文本数据大多属于非结构化的数据，要想对文本数据进行传统的统计模型分析，必须要经过层层的数据清洗与整理。

今天我们要介绍的『正则表达式及R字符串处理』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。如果把建立模型类比于高级厨师的工作，那么，数据清洗无疑是类似切菜洗碗打扫卫生的活儿。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，这种工作极有可能将占据你整个建模流程的80%的时间。

如果我们能够掌握高效的数据清洗工具，那么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，我举几个我自身经历的『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 (http://yphuang.github.io/blog/2016/03/09/R-Handling-and-Processing-Strings-Check-Attendance/)
- R辅助SAS处理大量数据：深入理解SAS之批量数据导入 (http://yphuang.github.io/blog/2016/03/03/Uderstanding-SAS-Import-Data-In-Batch/)
- R爬虫获取研究数据：这个将在后续的讲座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拓展包 `stringr` ，并接着介绍一些文件编码处理相关的函数。最后，通过一两个案例展示字符串处理的真实应用场景。

1.A toy example ——初步认识R中的字符串处理

为了先给大家一个关于R字符串处理的大体认识，我们使用R中自带的一个数据集 `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

```
# get number of characters in each state name

state_chars <- nchar(states)

# hist
hist(nchar(states),main = "Histogram",
     xlab = "number of charaters in US State names")


# longest state's name
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

```
# get states names with 'w'
grep(pattern = "w", x = states, value = TRUE)

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

# get states names with 'W' OR 'w'

## Method 1:
grep(pattern = "[wW]", x = states, value = TRUE)

## Method 2:
grep(pattern = "w", x = tolower(states),value = TRUE)

## Method 3:
grep(pattern = "W", x = toupper(states), value = TRUE)

## Method 4:
grep(pattern = "w",x = states, ignore.case = TRUE, value = TRUE)
```

字符统计：某些字母个数统计

```
library(stringr)

# total number of a's
str_count(states,"a")

###################
# number of vowels

# vector of vowels
vowels <- c("a","e","i","o","u")

# vector for storing results
num_vowels <- vector(mode = "integer",length = 5)

# calculate
for(i in seq_along(vowels)){
  num_aux <- str_count(tolower(states),vowels[i])
  num_vowels[i]<-sum(num_aux)
}

# add names
names(num_vowels)<-vowels

# total number of vowels
num_vowels

# barplot
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，则学习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` 包则已经能够游刃有余。

下面，我们先简要介绍重要并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他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。他们分别是：

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

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。正确理解他们的作用与用法，至关重要。

```
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.aslo.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.aslo.uses.regular.expressions", split=".")

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

#################
# function 2:
library(stringr)
str_extract_all(string = "my cridit card number: 34901358932236",pattern = "\\d")
```

- ^：匹配字符串的开始.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”以外的任何字符。

```
# function 1
test_vector<-c("123","456","321")
library(stringr)
str_extract_all(test_vector,"3")
str_extract_all(test_vector,"^3")

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

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

```
# function 1
test_vector<-c("123","456","321")
library(stringr)
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("AlphaGo实在厉害！","alphago是啥","阿尔法狗是一条很凶猛的狗。")
str_extract_all(string = test_vector2, pattern ="AlphaGo|阿尔法狗")
```

- ?：前面的字符(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")
```

- *：前面的字符(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)+")
```

- （）：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")
```

重复

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.aslo.uses.regular.expressions", split=".")

# compare
strsplit(x="strsplit.aslo.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:]	数字：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

[:xdigit:]	所有十六进制数字
—	—
[:punct:]	标点符号，如 . ， ； 等
[:graph:]	Graphical characters,即[:alnum:]和[:punct:]
[:blank:]	空字符，即：Space和Tab
[:space:]	Space，Tab，newline，及其他space characters
[:print:]	可打印的字符，即：[:alnum:]，[:punct:]和[:space:]

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

代表字符组的特殊符号

代码	含义说明
\w	字符串，等价于 [:alnum:]
\W	非字符串，等价于 [^[:alnum:]]
\s	空格字符，等价于 [:blank:]
\S	非空格字符，等价于 [^[:blank:]]
\d	数字，等价于 [:digit:]
\D	非数字，等价于 [^[:digit:]]
\b	Word edge（单词开头或结束的位置）
\B	No Word edge（非单词开头或结束的位置）
\<	Word beginning（单词开头的位置）
\>	Word end（单词结束的位置）

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_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。

```
# 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

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

###################
## 字符统计

# nchar
nchar(x)

# str_count
library(stringr)
str_count(x,pattern = "")
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

library(stringr)
str_replace_all(string = DNA,pattern = "Tt",replacement = "Uu")
```

字符串连接

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

# str_c()
library(stringr)
str_c("control",1:3,sep = "_")
```

字符串拆分

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

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

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。

```
###################################
## 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（请注意输出结果中world的首字母），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 = "")


#########################################
## 指定位置替换
```

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。

```
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 = "#.+#")

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

# str_extract_all，返回的列表中的元素为向量
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。

```
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()不对文本直接切割成向量，而是在文本内容中插入了缩进或分行的标识符。

```
string <- "Each character string in the input is first split into\n paragraphs (or lines containing whitespace only). The paragraphs are then formatted by breaking lines at word boundaries."

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()	查看编码格式；或者指定编码格式
tau::is.locale()	tests if the components of a vector of character are in the encoding of the current locale
tau::is.ascii()	
tau::is.utf8()	tests if the components of a vector of character are true UTF-8 strings

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。文锋写了一个相应的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 (http://qinwenfeng.com/cn/checkenc/)

```
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 (http://xccds1977.blogspot.sg/2012/04/r_12.html)，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。

```
library(stringr)
library(dplyr)

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
setInternet2()
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_repl
ace_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.numeri
c(.)

#获取评价分数
Score_line<-str_extract_all(string = web, pattern = '<span class="rating_num" propert
y="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 (https://en.wikibooks.org/wiki/R_Programming/Text_Processing)
- Introduction to stringr (https://cran.r-project.org/web/packages/stringr/vignettes/stringr.html)

- 正则表达式30分钟入门教程 (http://deerchao.net/tutorials/regex/regex.htm)
- 深入浅出之正则表达式 (http://dragon.cnblogs.com/archive/2006/05/08/394078.html)

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 (http://rstudio-pubs-static.s3.amazonaws.com/13823_dbf87ac4114b44f8a4b4fbd2ea5ea162.html)
- checkenc - 自动文本编码识别 (http://qinwenfeng.com/cn/checkenc/)
- 在R语言中使用正则表达式 (http://xccds1977.blogspot.sg/2012/04/r_12.html)