

Text Mining in R

文字處理 與 正規化

Pei Hsuan, Huang

Dec. 13, 2017

R 語言簡介

- R語言自1993年問世，用於統計分析、繪圖、資料採礦、矩陣運算與機器學習等多個面向
- 兩大特色：免費下載、開放原始碼。
- 套件：ggmap, ggplot
- R Studio/R Pubs/GitHub
- [軟體安裝教學\(R & SQL\)](#)


字串處理

常用字符串處理function

- nchar
- grep
- grepl
- regexpr
- sub
- gsub
- substr
- paste
- strsplit

example

```
> address <- c("臺北市文山區指南路二段91~120號"  
+             , "臺北市大同區重慶北路一段61~90號"  
+             , "臺北市文山區指南路三段1~30號"  
+             , "臺北市文山區指南路二段45巷31~60號"  
+             , "臺北市內湖區民權東路六段90巷6弄1~30號"  
+             , "臺北市文山區興隆路四段1~30號")
```

- (1)算總字數、中文字字數、數字字數
- (2)將個地址的 區、路 取出來
- (3)算文字的幾種 
- (4)針對 大同區or指南路 的地址，將他們的地址中的數字取代成 X

nchar

計算字串長度

```
> address
```

```
[1] "臺北市文山區指南路二段91~120號" "臺北市大同區重慶北路一段61~90號"  
[3] "臺北市文山區指南路三段1~30號" "臺北市文山區指南路二段45巷31~60號"  
[5] "臺北市內湖區民權東路六段90巷6弄1~30號" "臺北市文山區興隆路四段1~30號"
```

```
> nchar(address)
```

```
[1] 18 18 16 20 22 16
```

grep

grep第一個參數是pattern, 第二個是data,

output為有符合這個pattern是第幾筆資料

```
> grep(pattern = "文山區", x = address)
```

```
[1] 1 3 4 6
```

```
> grep("指南路", address)
```

```
[1] 1 3 4
```

grep (2)

```
> address
```

```
[1] "臺北市文山區指南路二段91~120號" "臺北市大同區重慶北路一段61~90號"  
[3] "臺北市文山區指南路三段1~30號" "臺北市文山區指南路二段45巷31~60號"  
[5] "臺北市內湖區民權東路六段90巷6弄1~30號" "臺北市文山區興隆路四段1~30號"
```

```
> (a <- grep("指南路二段", address))
```

```
[1] 1 4
```

```
> address[a]
```

```
[1] "臺北市文山區指南路二段91~120號" "臺北市文山區指南路二段45巷31~60號"
```


grepl

與grep用法相似,差異在於其output是TRUE/FALSE

```
> grepl(pattern = "文山區", x = address)
```

```
[1] TRUE FALSE TRUE TRUE FALSE TRUE
```

```
> grepl("指南路", address)
```

```
[1] TRUE FALSE TRUE TRUE FALSE FALSE
```

```
> ## 當多個條件的話
```

```
> grepl(c("指南路|二段"), address)
```

```
[1] TRUE FALSE TRUE TRUE FALSE FALSE
```

grepl (2)

```
> address
```

```
[1] "臺北市文山區指南路二段91~120號" "臺北市大同區重慶北路一段61~90號"  
[3] "臺北市文山區指南路三段1~30號" "臺北市文山區指南路二段45巷31~60號"  
[5] "臺北市內湖區民權東路六段90巷6弄1~30號" "臺北市文山區興隆路四段1~30號"
```

```
> (a <- grepl("指南路二段", address))
```

```
[1] TRUE FALSE FALSE TRUE FALSE FALSE
```

```
> address[a]
```

```
[1] "臺北市文山區指南路二段91~120號" "臺北市文山區指南路二段45巷31~60號"
```

regexpr

找出第一個符合pattern的字串在哪個位置及長度,

如果不符合pattern,會顯示-1

```
> address
```

```
[1] "臺北市文山區指南路二段91~120號" "臺北市大同區重慶北路一段61~90號"  
[3] "臺北市文山區指南路三段1~30號" "臺北市文山區指南路二段45巷31~60號"  
[5] "臺北市內湖區民權東路六段90巷6弄1~30號" "臺北市文山區興隆路四段1~30號"
```

```
> regexpr(pattern = "指", text = address)
```

```
[1] 7 -1 7 7 -1 -1  
attr(,"match.length")  
[1] 1 -1 1 1 -1 -1
```

regexpr (2)

```
> regexpr(pattern = "指南路", text = address)
```

```
[1]  7 -1  7  7 -1 -1  
attr(,"match.length")  
[1]  3 -1  3  3 -1 -1
```

```
> regexpr("指南路二段", address)
```

```
[1]  7 -1 -1  7 -1 -1  
attr(,"match.length")  
[1]  5 -1 -1  5 -1 -1
```

regexpr (3)

```
> regexpr("號", address)
```

```
[1] 18 18 16 20 22 16  
attr(,"match.length")  
[1] 1 1 1 1 1 1
```

```
> regexpr("[0-9]", address)
```

```
[1] 12 13 12 12 13 12  
attr(,"match.length")  
[1] 1 1 1 1 1 1
```

gregexpr

找出所有符合pattern的字串在哪個位置及長度

```
> address
```

```
[1] "臺北市文山區指南路二段91~120號" "臺北市大同區重慶北路一段61~90號"  
[3] "臺北市文山區指南路三段1~30號" "臺北市文山區指南路二段45巷31~60號"  
[5] "臺北市內湖區民權東路六段90巷6弄1~30號" "臺北市文山區興隆路四段1~30號"
```

先回憶一下剛剛regexpr的結果

```
> regexpr("[0-9]", address)
```

```
[1] 12 13 12 12 13 12  
attr(,"match.length")  
[1] 1 1 1 1 1 1
```

```
> gregexpr("[0-9]", address)
```

```
[[1]]
```

```
[1] 12 13 15 16 17
```

```
attr(,"match.length")
```

```
[1] 1 1 1 1 1
```

```
[[2]]
```

```
[1] 13 14 16 17
```

```
attr(,"match.length")
```

```
[1] 1 1 1 1
```

```
[[3]]
```

```
[1] 12 14 15
```

```
attr(,"match.length")
```

```
[1] 1 1 1
```

```
[[4]]
```

```
[1] 12 13 15 16 18 19
```

```
attr(,"match.length")
```

```
[1] 1 1 1 1 1 1
```

```
[[5]]
```

```
> matches = gregexpr("[\u4E00-\u9FA5]", address)
> (sent <- regmatches(address, matches) %>% sapply(function(x) paste0(x, collapse = ' ')))
```

```
[1] "臺北市文山區指南路二段號"      "臺北市大同區重慶北路一段號"
[3] "臺北市文山區指南路三段號"      "臺北市文山區指南路二段巷號"
[5] "臺北市內湖區民權東路六段巷弄號" "臺北市文山區興隆路四段號"
```



sub

sub指substitute,

把每個字串中第一個符合pattern的內容取代

```
> sub(pattern = "指南路", replacement = "AAA", x = address)
```

| | |
|------------------------------|------------------------|
| [1] "臺北市文山區AAA二段91~120號" | "臺北市大同區重慶北路一段61~90號" |
| [3] "臺北市文山區AAA三段1~30號" | "臺北市文山區AAA二段45巷31~60號" |
| [5] "臺北市內湖區民權東路六段90巷6弄1~30號" | "臺北市文山區興隆路四段1~30號" |

```
> sub("[0-9]", "X", address)
```

| | |
|------------------------------|------------------------|
| [1] "臺北市文山區指南路二段X1~120號" | "臺北市大同區重慶北路一段X1~90號" |
| [3] "臺北市文山區指南路三段X~30號" | "臺北市文山區指南路二段X5巷31~60號" |
| [5] "臺北市內湖區民權東路六段X0巷6弄1~30號" | "臺北市文山區興隆路四段X~30號" |

gsub

把每個字串中所有符合pattern的內容取代

```
> gsub(pattern = "指南路", replacement = "AAA", x = address)
```

| | |
|------------------------------|------------------------|
| [1] "臺北市文山區AAA二段91~120號" | "臺北市大同區重慶北路一段61~90號" |
| [3] "臺北市文山區AAA三段1~30號" | "臺北市文山區AAA二段45巷31~60號" |
| [5] "臺北市內湖區民權東路六段90巷6弄1~30號" | "臺北市文山區興隆路四段1~30號" |

```
> gsub("[0-9]", "X", address)
```

| | |
|------------------------------|------------------------|
| [1] "臺北市文山區指南路二段XX~XXX號" | "臺北市大同區重慶北路一段XX~XX號" |
| [3] "臺北市文山區指南路三段X~XX號" | "臺北市文山區指南路二段XX巷XX~XX號" |
| [5] "臺北市內湖區民權東路六段XX巷X弄X~XX號" | "臺北市文山區興隆路四段X~XX號" |

substr

```
> address
```

```
[1] "臺北市文山區指南路二段91~120號" "臺北市大同區重慶北路一段61~90號"  
[3] "臺北市文山區指南路三段1~30號" "臺北市文山區指南路二段45巷31~60號"  
[5] "臺北市內湖區民權東路六段90巷6弄1~30號" "臺北市文山區興隆路四段1~30號"
```

- 如果我只想要地址中的[行政區 + 路段]

```
> substr(address, start = 4, stop = 12)
```

```
[1] "文山區指南路二段9" "大同區重慶北路一段" "文山區指南路三段1" "文山區指南路二段4"  
[5] "內湖區民權東路六段" "文山區興隆路四段1"
```

strsplit

字串的切割

```
> strsplit(address, "市")
```

```
[[1]]  
[1] "臺北"          "文山區指南路二段91~120號"
```

```
[[2]]  
[1] "臺北"          "大同區重慶北路一段61~90號"
```

```
[[3]]  
[1] "臺北"          "文山區指南路三段1~30號"
```

```
[[4]]  
[1] "臺北"          "文山區指南路二段45巷31~60號"
```

```
[[5]]  
[1] "臺北"          "內湖區民權東路六段90巷6弄1~30號"
```

正規表示法

資料

什麼是「正規化表示」？

Wiki：正規表示式使用單個字串來描述、符合一系列符合某個句法規則的字串。在很多文字編輯器裡，正則運算式通常被用來檢索、替換那些符合某個模式的文字。

Bacon：跨程式的語言"規則"、以更精簡的方式描述語言

```
> set.seed(2)
>
> data = NULL
> for(i in 1:10){
+   tem = sample(letters[1:3], 20, replace = T) %>% paste0(collapse = '')
+   data = c(data, tem)
+ }
>
> data %>% head(3)
```

```
[1] "acbccacbbbacabccaba" "bbcabbabcaaaccbbcaca" "caaaccbbcaaccacccbc"
```

最常見的正規化表示法

正規劃通常是用在「檢查」、「搜尋」文字字串上

```
> gsub('[:punct:]', '', '政大統研8+9')
```

```
[1] "政大統研89"
```

```
> gsub('[:digit:]', '', '政大統研8+9')
```

```
[1] "政大統研+"
```

正規化常見指令

檢查字串裡是否有 a

```
> pattern = 'a'  
> data[grep(pattern, data)]
```

```
[1] "acbaccacbbbacabccaba" "bbcabbabcaaaccbbcaca" "caaaccbbcaaccacccbc" "ccbacbbbbaaaaaaacacbb"  
[5] "bcabaccacbcbbbabaab" "abccabcbaaacbbcccacb" "aaccacbacbacacabbcab" "aababccabaabcbcacaaa"  
[9] "ccacabacabcaacbcabb" "cbccacacaaabcabaaaaa"
```

檢查字串裡是否有 aa

檢查字串裡是否有 aaa.... (連續三個a重複出現)

```
> pattern = 'a{3,}'  
> data[grep(pattern, data)] %>% head(3)
```

```
[1] "bbcabbabcaaaccbbcaca" "caaaccbbcaaccacccbc" "ccbacbbbbaaaaaaacacbb"
```


檢查字串是否 a 開頭

```
> pattern = '^a'  
> data[grep(pattern, data)]
```

```
[1] "acbaccacbbbacabccaba" "abccabcbaaacbbcccacb" "aaccacbacbacacabbcab" "aababccabaabcbcacaaa"
```

檢查字串是否 a 結尾

```
> pattern = 'a$'  
> data[grep(pattern, data)]
```

```
[1] "acbaccacbbbacabccaba" "bbcabbabcaaaccbbcaca" "aababccabaabcbcacaaa" "cbccacacaaabcabaaaaa"
```

檢查字串是否 a 開頭, a結尾



```
> pattern = '^a(.)*a$'  
> data[grep(pattern, data)]
```

```
[1] "acbaccacbbbacabccaba" "aababccabaabcbcacaaa"
```

25/29

`*`, `+`, `?` 的應用


- `'*'` : 匹配0次至無限次
- `'+'` : 匹配1次至無限次
- `'?'` : 匹配0次至一次

[]

- '[' : 將要檢查的字元放在中括號內，只要在裡面的字串都會被檢查出來, e.g. [0-9] 是要檢查字串中是否有 0 ~ 9 的元素
 - : 在中括號裡的 ^ 是「非」的意思

```
> gsub(pattern = '[a-e]', 'X', 'bacon')
```

```
[1] "XXXon"
```

```
>  
> grepl(pattern = 'r\\wg', 'regex') 
```

```
[1] TRUE
```

[a-z.]+ 可以匹配 "test.i.ng"

完整正規劃表示法 的匹配

```
> library(stringr)
> files = c("block010_dplyr-end-single-table.rmd", "testing.txt", "vlock02_spd.rmd")
> pattern <- "^block\\d{3}_.*dplyr-(.*)\\.rmd$"
> (na.omit(str_match(files, pattern)))
```



```
      [,1]      [,2]
[1,] "block010_dplyr-end-single-table.rmd" "end-single-table"
attr(,"na.action")
[1] 2 3
attr(,"class")
[1] "omit"
```

Reference

- [正規表示式 Regular Expression](#)
- [Stringr套件](#)