Introducción a la adaptación y modificación de cadenas en R usando Expresiones regulares

Trabajar con datos estadísticos en R implica una gran cantidad de datos de texto o cadenas de caracteres, incluido el ajuste de los nombres de las variables exportadas al formato del nombre de la variable R, el cambio de los niveles de las variables categóricas, el procesamiento de los datos de texto para su uso en LaTeX. Estas tareas se pueden realizar mediante el uso de funciones para la búsqueda de cadenas (o coincidencia) y la modificación

R proporciona varias de estas funciones. Los más utilizados son `grep()`, `gsub()`, `strsplit()`. Al usarlas, es importante saber que algunos de sus argumentos son interpretados por R como expresiones regulares.

¿Qué es una expresión regular? Según la ayuda de Linux, una expresión regular es un patrón que describe un conjunto de cadenas. En pocas palabras, la expresión regular es una "instrucción" dada a una función acerca de qué y cómo hacer coincidir o reemplazar cadenas. El uso de expresiones regulares puede resolver problemas complicados (no todos los problemas) en el emparejamiento y manipulación de cadenas, y puede reducir el tiempo empleado en la escritura y el mantenimiento del código R.

Cómo NO usar expresiones regulares: Cuidado con los metacaracteres

Las funciones de coincidencia y modificación de cadenas R interpretan algunos de sus argumentos como expresiones regulares. Por ejemplo, el patrón de argumento de la función `gsub()` es una cadena de caracteres interpuesta como una expresión regular. Un usuario puede no ser consciente de que obtiene un error o de que no realiza su tarea, no lo nota. Por ejemplo, queremos sustituir una "$" con un punto en la cadena s usando la función `gsub()`.

```
s = "gsub$uses$regular$expressions"
```

Nuestro resultado final debe ser la cadena `s1`.

```
s1 = "gsub.uses.regular.expressions"
```

Si no sabemos que `gsub()` trata el argumento "pattern" como una expresión regular,

La cadena `s1` no es lo que queríamos porque `gsub()` interpretó el carácter "$" como un carácter especial de expresiones regulares. Para obtener el resultado correcto, tenemos que decirle a `gsub()` que interprete "$" como un carácter normal. Esto se puede hacer precediendo "$" con una doble barra invertida. La solución correcta es

```
s1 = gsub(pattern = "\\$",
     replacement = ".", "gsub$uses$regular$expressions")

s1
[1] "gsub.uses.regular.expressions"
```

En expresiones regulares, los caracteres

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

se llaman metacaracteres. Para hacer coincidir cualquier un metacarácter como un carácter normal, debe ir precedido de una doble barra hacia atrás. Para hacer coincidir una barra hacia atrás con un carácter normal, se escriben cuatro barras invertidas (vea los ejemplos a continuación).

Ejemplos de uso no intencional de expresiones regulares que resultan en errores

```
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=".")
```

Ejemplos de formas correctas de evitar el uso de expresiones regulares

```
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="\\.")
strsplit(x="strsplit.aslo.uses.regular.expressions", split=".",
         fixed=TRUE)
```

Supongamos que exportamos un bastidor de datos desde un archivo de texto usando la

```
   id...of....patient      patient....age
1                    1                   1
2                    2                   2
```

Nos gustaría reemplazar los puntos múltiples por uno solo. Primero, nos deshacemos de los 4 puntos y luego de los 3 puntos.

```
names(d1) = gsub(pattern = "\\.\\.\\.\\.", replacement = ".",
                 names(d1))
names(d1)
```

```
[1] "id...of.patient" "patient.age"
```

```
names(d1) = gsub(pattern = "\\.\\.\\.", replacement = ".",
                 names(d1))
names(d1)
```

```
[1] "id.of.patient" "patient.age"
```

La principal desventaja de esta solución es que cuando los datos cambian, tenemos que volver a escribir el código para los nuevos datos, lo que va en contra del espíritu de la programación eficiente. Queremos minimizar el tiempo dedicado al mantenimiento del código y además tiempo libre para el análisis de datos. Aquí es donde las expresiones regulares vienen al rescate. La siguiente solución siempre funcionará, incluso si los nombres de la variable cambian.

```
names(d1) <- gsub(pattern = "\\.+", replacement = ".",
                  x = names(d1))
names(d1)
```

```
[1] "id.of.patient" "patient.age"
```

La expresión regular

```
\\.+
```

le dice a la función `gsub()` que haga coincidir y reemplace una o más repeticiones de un punto. El metacarácter "+" es la instrucción para hacer coincidir una o más repeticiones de lo que viene antes de "+" .

He aquí hay algunos metacaracteres y sus significados.

"." coincide con todo excepto la cadena vacía "".
"+" el elemento anterior coincidirá una o más veces.
"*" el elemento anterior se comparará cero o más veces.

"(", ")" paréntesis para agrupar.
"[", "]" corchetes de la clase caracter.

La clase carácter a nuestro servicio

Consideremos la tarea de reemplazar todos los valores que no contengan letras o dígitos por un valor de NA. He aquí hay una forma de lograr esto sin usar expresiones regulares.

```
d = data.frame(id = c(11, 22, 33, 44, 55, 66, 77, 88),
               drug = c("vitamin E","vitamin ESTER-C",
               " vitamin E ", "vitamin E(ointment)","",
               "vitamin E ", "provitamin E\n", "vit E"),
               text = c("","  ", "3 times a day after meal",
               "once a day", " "," one per day ", "\t", "\n "),
               stringsAsFactors = FALSE)
s = d$text
unique(s)
```

```
[1] "" " "
[3] "3 times a day after meal" "once a day"
[5] " " " one per day "
[7] "\t"
```

```
s[s == "" | s == " " | s == " " | s == "\t" | s == "\n"] = NA
```

Como en el caso anterior, esta solución no es buena porque cuando los datos cambian, el código también debe cambiarse. Además, los datos pueden ser tan grandes que es imposible (toma demasiado tiempo) verificarlos manualmente. Veamos cómo podemos emplear expresiones regulares en este caso.

Debemos decir a la función `grep()` que solo necesitamos cadenas que contengan letras o dígitos. Las expresiones regulares nos permiten hacerlo describiendo un conjunto de caracteres. Dicho conjunto se denomina clase de caracteres y se denota entre corchetes. Por ejemplo, la expresión regular "[nco]" coincide con cualquier cadena que contenga cualquiera de los caracteres "n", "c", "o".

```
grep("[nco]", c("nose", "letter38", "window9", "apple0"),
     value = TRUE)
```

```
[1] "nose" "window9"
```

Otro ejemplo de una clase de caracteres es la expresión regular "[01234567]" o "[0-7]" coincide con cualquier cadena que contenga cualquiera de los dígitos de "0" a "7".

```
grep("[01234567]", c("nose", "letter38", "window9", "apple0"),
```

"[0-9]" - Dígitos
"[a-z]" – Letras minúsculas
"[A-Z]" – Letras mayúsculas
"[a-zA-Z]" – Caracteres alfabéticos
"[^a-zA-Z]" – Caracteres no alfabéticos
"[a-zA-Z0-9]" – Caracteres alfanuméricos
"[\t\n\r\f\v]" – Caracteres de espaciado
"[]$*+.?[^{|(\\#%&~_/<=>'!,:;`\")}@-]" – Caracteres de puntuación

Para resolver nuestro problema utilizamos clase de caracteres alfanuméricos.

```
s = d$text; s

[1] "" " "
[3] "3 times a day after meal" "once a day"
[5] " " " one per day "
[7] "\t"

allIndices = 1:length(s)
letOrDigIndices = grep("[a-zA-Z0-9]", s)
blankInd = setdiff(allIndices, letOrDigIndices)
s[blankInd] = NA; s

[1] NA NA
[3] "3 times a day after meal" "once a day"
[5] NA " one per day "
[7] NA NA
```

Esta solución es mejor que la anterior, ya que, si nuestros datos cambian, no tenemos que cambiar nuestro código.

Otra forma de hacer esto:

```
s = d$text
s = gsub("^$|^( +)$|[\t\n\r\f\v]+", NA, s); s

[1] NA NA
[3] "3 times a day after meal" "once a day"
[5] NA " one per day "
[7] NA
```

Si desea deshacerse de espacios en blanco adicionales, el siguiente código es útil.

```
s = d$text
s = gsub("^([ \t\n\r\f\v]+)|([ \t\n\r\f\v]+)$", "", s); s

[1] "" ""
[3] "3 times a day after meal" "once a day"
```

Cuidado con los caracteres especiales dentro de una clase de personajes

Hemos aprendido que para hacer coincidir un metacarácter con un carácter normal, debemos precederlo con una doble barra hacia atrás. Esta regla no se mantiene dentro de la clase carácter; casi todos los metacaracteres pierden su poder mágico con los siguientes caracteres.

```
] ^ - \
```

Aquí hay un conjunto de reglas sobre cómo hacer coincidir los caracteres como caracteres normales dentro de la clase carácter.

Para coincidir "]" dentro de una clase de caracteres, ponerlo primero.

Para unir "-" dentro de una clase de caracteres, ponerlo primero o último

Para coincidir con "^" dentro de una clase de caracteres, colocarlo en cualquier lugar, excepto el primero.

Para hacer coincidir cualquier otro carácter o metacarácter (excepto \) dentro de una clase de caracteres, colóquelo en cualquier lugar.

Ejemplo útil

Tenemos aquí un ejemplo útil, en el que podemos practicar el uso de clases de caracter y metacaracteres.

Nos gustaría hacer coincidir todas las cadenas que contienen "vitamina E". Sin utilizar expresiones regulares, es muy fácil perder coincidencias relevantes o hacer coincidir algo que no necesitamos.

```
s = d$drug; s

[1] "vitamin E" "vitamin ESTER-C" " vitamin E "
[4] "vitamin E(ointment)" "" "vitamin E "
[7] "provitamin E\n" "vit E"

grep("vitamin e", s, ignore.case = TRUE, value = TRUE)

[1] "vitamin E" "vitamin ESTER-C" " vitamin E "
[4] "vitamin E(ointment)" "vitamin E " "provitamin E\n"
```

Usar expresiones regulares ayuda a refinar nuestra búsqueda.

Primero, excluimos todas las cadenas con "vitamina e" seguida de una letra o nada (una

Las expresiones regulares como "vitamina e ($ | [^ a-zA-Z])" le dicen a R que encuentre todas las cadenas de caracteres "vitamina e" seguidas de un no carácter (grupo de caracteres [^ a-zA-Z]) o una cadena vacía (metacarácter "$"). Aquí también utilizamos corchetes para asegurarnos de que el operador "o" (metacarácter "|") funcione en las partes correctas de nuestra expresión regular.

En segundo lugar, incluimos la cadena "vit e" en nuestra búsqueda utilizando el operador "o" (metacarácter "|").

```
grep("vitamin e($|[^a-zA-Z])|vit e($|[^a-zA-Z])", s,
     ignore.case = TRUE,value = TRUE)
```

```
[1] "vitamin E" " vitamin E " "vitamin E(ointment)"
[4] "vitamin E " "provitamin E\n" "vit E"
```

Si deseamos deshacernos de "provitamin" usamos lo siguiente:

```
grep("([^a-z]+|^)vitamin e($|[^a-zA-Z])|([^a-z]+|^)vit e($|[^a-
zA-Z])", s, ignore.case = TRUE, value = TRUE)
```

```
[1] "vitamin E" " vitamin E " "vitamin E(ointment)"
[4] "vitamin E " "vit E"
```

Se puede conseguir algo realmente imaginativo:

```
grep("([^a-z]+|^)vitamin([^a-zA-Z]+)e($|[^a-zA-Z])|([^a-
z]+|^)vit([^a-zA-Z]+)e($|[^a-+ s,
ignore.case = TRUE, value = TRUE)
```

```
[1] "vitamin E" " vitamin E " "vitamin E(ointment)"
[4] "vitamin E " "vit E"
```