

Introducción a regex con rebus

Periodismo de datos
Febrero, 2021

M.C. JORGE JUVENAL CAMPOS FERREIRA.
Investigador Asociado.
Laboratorio Nacional de Políticas Públicas
CIDE

Introducción

Requerimientos para la sesión:

Instalar los paquetes {rebus}, {stringr} y {htmltools}.

Introducción

Como vimos la clase pasada con Sebastián, las funciones de la librería stringr funcionan con algo llamado “patrones”.

```
str_view(string, pattern, match = NA)  
str_view_all(string, pattern, match = NA)
```

Sin embargo... ¿qué son y cómo podemos generar esos patrones?

Para eso, en programación contamos con una herramienta muy poderosa conocida como *regex* o *expresiones regulares*.

Expresiones regulares



Las expresiones regulares son **secuencias de caracteres** para definir un patrón de búsqueda.

Las expresiones regulares sirven **tanto para quedarnos** con palabras/frases importantes, así **como para deshacernos de ellas.**

Inversión de tiempo que todo buen analista de datos tiene que hacer.
Son la navaja suiza para trabajar con datos de texto.

¿Cómo funcionan las expresiones regulares?



Regex



¿Cómo se ven las expresiones regulares?

Las regex son cadenas de texto que, a través de símbolos predefinidos, nos sirven para detectar un patrón en un texto.

Regex



¿Cómo se ven las expresiones regulares?

Las regex son cadenas de texto que, a través de símbolos predefinidos, nos sirven para detectar un patrón en un texto.

Ejemplo:

```
(?<![\\w\\d])Retweet(?:[\\w\\d])(\\s[\\d\\. (K|M)?]+)?
```

(Regex para capturar el numero de retweets de una base de datos en TW)

Regex



¿Cómo se ven las expresiones regulares?

Las regex son cadenas de texto que, a través de símbolos predefinidos, nos sirven para detectar un patrón en un texto.

Ejemplo:

```
(?<![\\w\\d])Retweet(?:[\\w\\d])(\\s[\\d\\. (K|M)?]+)?
```

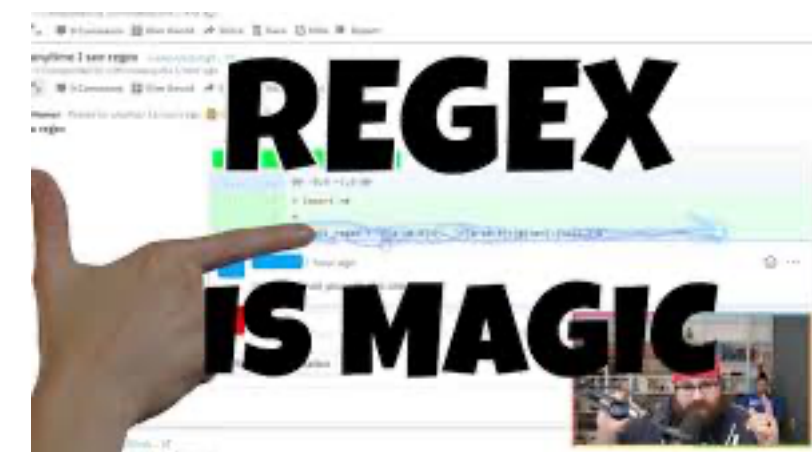
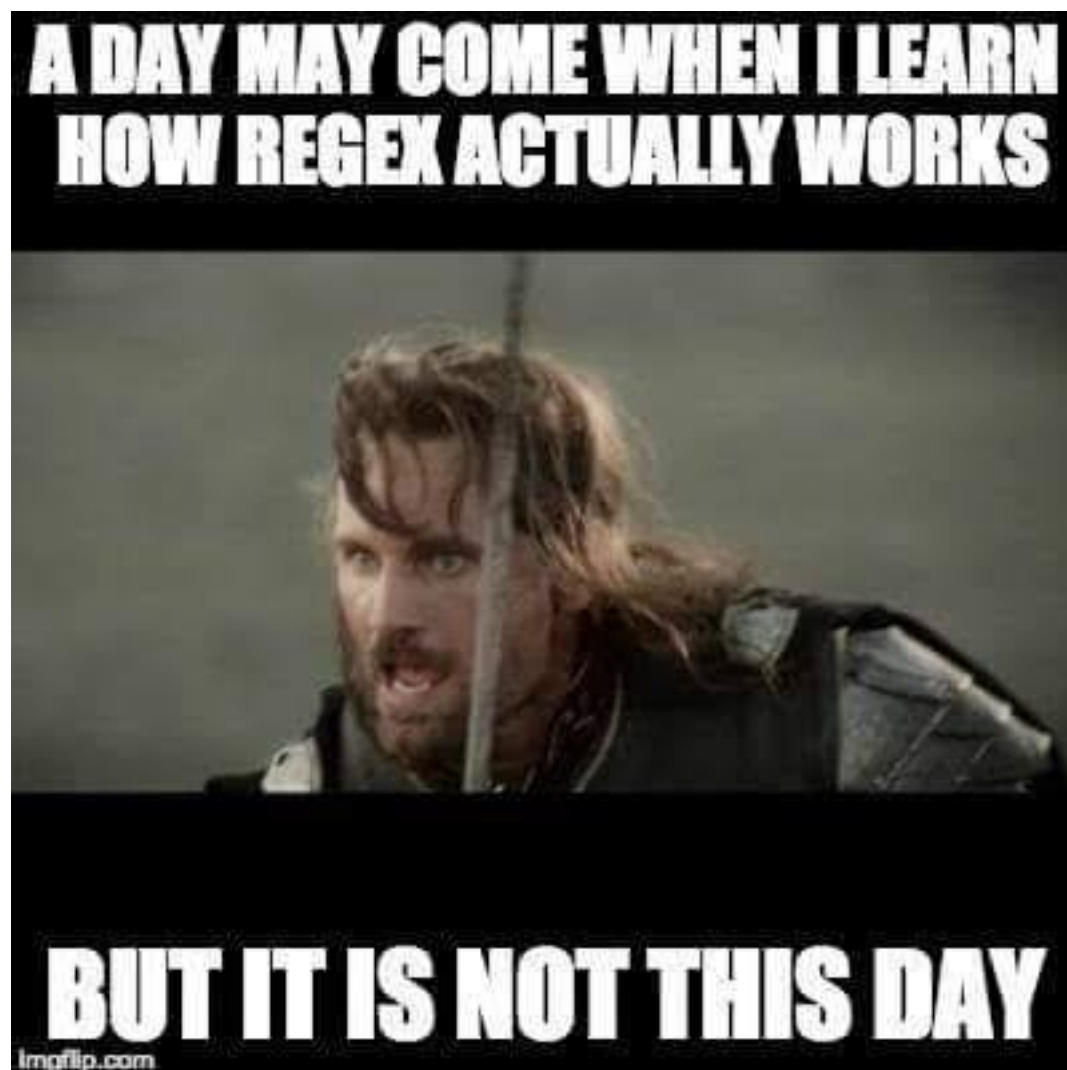
(Regex para capturar el numero de retweets de una base de datos en TW)



Regex



Las regex son conceptos complicados (incluso para los programadores), así que tómenlo con calma. 😊



{rebus}

En R, existe una forma menos dolorosa para poder aprender (y utilizar) estos conceptos sin tanto sufrimiento.

La librería {rebus} nos **da la facilidad de construir expresiones regulares de manera más intuitiva**, mientras vamos aprendiendo a trabajar con las *regex* tradicionales.

rebus: Build Regular Expressions in a Human Readable Way

Build regular expressions piece by piece using human readable code. This package is designed for interactive use.

Documentación:

<https://www.rdocumentation.org/packages/rebus/versions/0.1-3>

Objetos pre-programados {rebus}

Patrón	Expresión Regular	Objeto rebus
Inicio de un string o cadena de texto	^	START
Final de un string	\$	END
Cualquier carácter sencillo	.	ANY_CHAR
Punto literal, gorrito o signo de pesos	\. \^ \\$	DOT, CARAT, DOLLAR

Objetos pre-programados {rebus}

Objeto Rebus	Regex Tradicional	Interpretación
UPPER	[::upper:]	Letras en Mayúsculas
PUNCT	[::punct:]	Signo de puntuación
DOT	\\.	Punto

Otras Funciones

Patrón	Expresión Regular	Función rebus
Valor exacto	^\$	exactly()
Captura	()	capture()
Este patrón o este patrón (varios en uno)	(?:a b)	or1()

Funciones de repetición {rebus}

Patrón	Expresión Regular	Función rebus
Opcional	?	optional()
Zero o más	*	zero_or_more()
Uno o más	+	one_or_more()
Entre <i>n</i> y <i>m</i> veces	{n}{m}	repeated()

Caracteres especiales

Cuando estamos generando expresiones regulares en R, tenemos que tener cuidado al utilizar los símbolos siguientes:

- Paréntesis: “(“ y “)”
- Corchetes: “[“ y “]”
- Gorritos: “^”
- Símbolos de Moneda “\$”
- Guiones: “-“ o “_”
- Símbolo de Más: “+”
- Símbolo de interrogación: “?”

Si queremos diseñar patrones utilizando estos símbolos especiales, tenemos que “escaparlos” primero, utilizando “\\”, por ejemplo:

```
pat <- "\\[" %R% capture(one_or_more(DGT)) %R% "\\]"
```

Función `char_class()`

Esta función sirve para **definir un conjunto de caracteres que van a formar parte del patrón.**
Por ejemplo:

```
library(rebus)
c <- char_class("aeiouAEIOUñ@")
str_view_all("Estos niñ@s son mis Alumnos", pattern = c)
```

En este caso, creamos un objeto en el cual el patrón a detectar va a ser todas las vocales, minúsculas y mayúsculas, la letra ñ y el arroba. Abajo, podemos ver lo que captura este patrón:

Estos niñ@s son mis Alumnos

Función %R% (pipa rebus, concatenar)

Esta función sirve **para concatenar objetos rebus**, para poder armar patrones compuestos.

Ejemplo de uso:

```
pat <- START %R% WRD %R% DGT %R% capture(one_or_more(SPC)) %R% END
```

Patrón compuesto.

Este patrón va a capturar:

“Al inicio del renglón”: “una letra” + “un dígito” + “uno o varios espacios” + “final del string”

Funciones:



`stringr::str_view(string, pattern, match)` `stringr::str_view_all(string, pattern, match)`

Esta función nos permite probar nuestros intentos de expresiones regulares. “Para atrapar lo que queremos atrapar.”



Atrapa a todos los números de manera individual

```
str_view_all(contact, pattern = DGT , match = TRUE)
```

Solo se visualizan donde hay coincidencias con el patrón

Objeto rebus de patrón de dígito o números

Call me at 555-555-0191

123 Main St

(555) 555 0191

Phone: 555.555.0191 Mobile: 555.555.0192

Manos a la obra

A continuación vamos a llevar a cabo un ejemplo

Manos a la obra

1. Leemos las librerías

```
library(rebus)  
library(stringr)
```

2. Generamos texto

```
# Some strings to practice with  
x <- c("cat", "coat", "scotland", "tic toc")
```

3. Primer patrón!

Generamos un patrón de las palabras que empiezan con la letra “c”

```
# Run me  
str_view(x, pattern = START %R% "c")
```

Ejemplos

4. Resultado

Se marca en oscuro la letra c inicial.

```
# Run me  
str_view(x, pattern = START %R% "c")
```

cat

coat

scotland

tic toc

Ejemplos

5. Ahora, las que terminen en “-at”

```
# Match the strings that end with "at"  
str_view(x, pattern = "at" %R% END)
```

cat

coat

scotland

tic toc

Ejemplos

6. Palabras que llevan un caracter, y luego llevan una “t”

```
x <- c("cat", "coat", "scotland", "tic toc")  
  
# Match any character followed by a "t"  
str_view(x, pattern = ANY_CHAR %R% "t")
```

cat

coat

scotland

tic toc

Ejemplos

7. Palabras de exactamente 3 caracteres.

```
# Match a string with exactly three characters  
str_view(x, pattern = START %R% ANY_CHAR %R% ANY_CHAR %R% ANY_CHAR  
%R% END)
```

cat

coat

scotland

tic toc

Ejercicio.



Abramos RStudio y corramos el ejemplo que les envié a su correo.