
quantil

Uso ingenioso de expresiones regulares

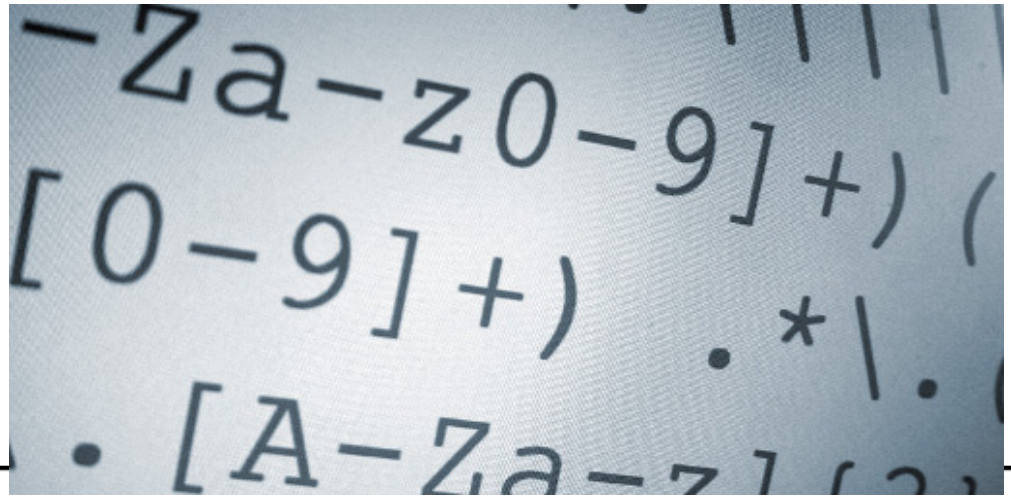
Francisco Barreras

27 de octubre de 2015

Introducción

¿Qué son expresiones regulares?

- Son la notación estándar para caracterizar secuencias de texto.
- Una expresión regular es un patrón de búsqueda que al aplicarse a una linea de texto, nos dice si contiene o no el patrón.
- Algunos editores de texto como **Word** las usan para encontrar o reemplazar palabras, pero sirven para búsquedas mucho más complejas.



Algunos ejemplos

Con expresiones regulares podemos hacer tareas como:

- Buscar todos los precios en un texto. Identificando todos los strings que sean como $1.999 o $1'000.000
- Buscar todas las palabras que empiecen por mayúscula en un texto.
- Identificar los URLs en un tweet o status de Facebook.
- Identificar las palabras que están entre una palabra de negación y un signo de puntuación.

Algunas definiciones

- **String:** Es una secuencia de símbolos alfanuméricos. El espacio se trata como un símbolo más y se denota como ␣.
- **Expresión Regular**: Es un patrón de búsqueda que define un conjunto de Strings. A saber, los strings con los que hace match.
- **Lenguaje generado:** Son todos los strings que hacen match con una expresión regular dada.
- **Corpus:** Es el conjunto de strings al que se le hace la búsqueda o reemplazo. Puede ser un documento, un tweet o una linea de texto.

¿Cómo opera una expresión regular?

Aplicación

Dependiendo del software que se utilice, una expresión regular puede retornar:

- **BOOLEAN:** Si el string contiene el patrón buscado.
- **INDEX:** Los índices de inicio y fin del patrón emparejado en el string.
- **PATRÓN:** Para búsquedas complejas, me puede interesar el patrón encontrado como tal.

En **R** el paquete base tiene funciones como **grep, grepl y gsub** que nos ayudan en estas tareas.

Expresiones regulares básicas

Patrón de búsqueda simple

Una expresión regular puede consistir de un sólo caracter o de varios caracteres. Para Pearl (El lenguaje que maneja expresiones regulares), las **mayúsculas** y las **minúsculas** son caracteres diferentes.

RE	Example Patterns Matched
/woodchucks/	"interesting links to woodchucks and lemurs"
/a/	"Mary Ann stopped by Mona's"
/Claire␣says,/	"Dagmar, my gift please," Claire says,"
/DOROTHY/	"SURRENDER DOROTHY"
/!/	"You've left the burglar behind again!" said Nori

Disjunción

- ¿Qué pasa si queremos buscar un patrón más amplio, como "quantil" o "Quantil"?
- En este caso necesitamos una **disjunción**.

Disjunción

Para indicar que queremos un caracter u otro, usamos el operador **[ ]**. Para buscar "quantil" o "Quantil" necesitamos la siguiente expresión regular:

/[Qq]uantil/

Disjunción

RE	Match	Example Patterns
`/[wW]oodchuck/`	Woodchuck or woodchuck	"Woodchuck"
`/[abc]/`	'a', 'b', *or* 'c'	"In uomini, in soldati"
`/[1234567890]/`	any digit	"plenty of 7 to 5"

Figure 2.1 The use of the brackets `[]` to specify a disjunction of characters.

Rangos

- Los rangos nos permiten abreviar algunas expresiones regulares.
- **/[2-7]/** busca todos los dígitos entre **2** y **7**.
- **/[A-G]/** busca todas las mayúsculas entre **A** y **G**.
- **/[a-fk-o/** busca todas las minúsculas entre la **a** y la **f**, y entre la **k** y la **o**.

Negación

¿Qué pasa si queremos **excluir** algunos caracteres de la búsqueda?

Negación

- Para negar incluímos el símbolo ^ al comienzo de los corchetes cuadrados **[]**.
- **/[^sS]/** encuentra lo que no sea "S" o "s".
- **/[^[A-Z]]/** retorna "no una letra mayúscula".

Cuantificadores

- A menudo queremos buscar distinto **número de ocurrencias** de un patrón.
- Los **Cuantificadores** nos ayudan a incorporar esta idea en nuestras expresiones regulares.
- Los símbolos **?,+,*,**$\{$**,**$\}$ son cuantificadores.

/ho+la/ empareja "hola","hoola" y "hooooooola".
/perros?/ empareja "perro" o "perros".

Cuantificadores

Cuantificadores

- **?** sirve para emparejar cero o una ocurrencia del caracter anterior.
- **+** se conoce como **Kleene plus** sirve para emparejar una o más ocurrencias del caracter anterior.
- ***** se conoce como **Kleene star** sirve para emparejas cero o más ocurrencias del caracter anterior.
- **{m,n}** sirve para emparejar entre m y n ocurrencias del caracter anterior.

Anclas

- A menudo queremos ubicar una expresión **sólo al comienzo** de una linea, o queremos especificar que una expresión esté antes de **acabar una palabra**.
- ¿Cómo buscamos la palabra "humo", sin encontrar también la palabra "humor"? ¿Cómo encontramos las lineas que **empiezen por un número** en la Biblia?
- Usamos **anclas**.

Ejemplo

/ \b humo\b /
/^[1-9]+/

Anclas

Anclas

- ^ al inicio de una expresión regular significa que va a buscar el patrón al inicio del string proporcionado.
- **$** al final de una expresión regular significa que va a buscar el patrón al final del string proporcionado.
- **b** se empareja con la frontera de una palabra.
- **B** se empareja con lo que **no** sea la frontera de una palabra.

`* Noten que a veces se usa el \ para distinguir los caracteres tradicionales de los operadores.`

Wildcards

- Los **wildcards** son otras herramientas que nos ayudan a ahorrar tiempo al escribir expresiones regulares y hacen todo más fácil.
- ¿Cómo buscamos todas las lineas que contengan dos veces la palabra Amor?

Wildcards

- **W** o *Whitespace* busca cualquier espacio, tabulación o enter en el string seleccionado.
- **w** o *Not whitespace* busca cualquier caracter alfanumérico.
- **.** busca cualquier caracter.

/Amor . * Amor/ es casi la expresión que buscamos.

Disjunción

Para buscar una u otra expresión regular, usamos el operador |.

- **/perros|gatos/** busca los patrones "perro" o "gato".
- ¿Qué pasa si en un texto quiero buscar los patrones "príncipe" o "princesa"?
- **/príncipe|esa/** va a emparejas "príncipe" o "esa".
- Necesitamos agrupar la subexpresión donde nos interesa hacer la disjunción.

/prínc(ipe|esa)/ es la expresión que buscamos.

Backtracking

A veces me interesa referirme a una expresión regular que encontré. Bien para **sustituir**, o para **buscar varias veces** algo indeterminado.

- Cuando envolvemos una expresión regular en paréntesis **( )** , esta queda almacenada en la primera vacante de memoria.
- Los siguientes paréntesis se guardan en las vacantes sucesivas.
- Para llamar a esas expresiones guardadas se usa \1, \2, etc.

Ejemplo

¿Cómo buscamos si un adverbio se repite en un texto?

/([a-z]+mente).* \1/ es la expresión que necesitamos.

Lenguaje regular generado

Dada una expresión regular, su lenguaje regular son todos los strings que son aceptados por esa expresión regular. Por ejemplo el **lenguaje de ovejas**:

/baa+!/

Corresponde al siguiente lenguaje regular:

- baa!
- baaa!
- baaaa!
- ...

Tutorial práctico

Tutorial

Una buena fuente de material para repasar este lenguaje es:

REGEXONE

Cosas más avanzadas

Lookforward y lookbehind

- Las expresiones regulares permiten hacer búsquedas en contextos.
- Posiblemente no sólo estoy interesado en un patrón, sino en un patrón que esté cerca de una palabra dada (pero sin incluírla).
- O puedo estar interesado en un patrón que NO esté entre comillas.
- ¿Qué tal un número que no esté antecedido por un signo pesos?

Lookarounds

- Los lookarounds son patrones de "longitud cero" al igual que el de frontera de palabra o inicio de línea.
- Estas expresiones emparejan algunos caracteres. Pero luego los descartan.
- No devuelven caracteres de un string, sólo dicen si el match es posible o no.
- Es importante entender que las expresiones regulares son un autómata finito, recorre el string caracter por caracter. Esto es clave en el funcionamiento de los lookarounds.

Lookaheads

Lookahead

*expr1(***?=***expr2)* hace match a expr1 sólo si inmediatamente después sigue expr2.

Negative Lookahead

*expr1(***?!***expr2)* hace match a expr1 sólo si inmediatamente después **no** sigue expr2.

Lookbehinds

Lookbehind

(?<=expr2)expr1 hace match a expr1 sólo si inmediatamente antes está expr2.

Negative Lookbehind

(?<!expr2)expr1 hace match a expr1 sólo si inmediatamente antes **no** está expr2.

¿Cómo funciona un lookbehind?

Veamos como funciona la expresión regular "(? <!ser)humano" sobre el string:

" Un humano"

- Se para en la U, retrocede 3 e intenta darle match a "ser", como falla continúa.
- Parado en la U intenta hacer match con la primera "h" falla y continúa.
- Se para en la "n" retrocede 3 e intenta darle match a "ser", como falla continúa.
- Parado en la n, intenta hacer match con la letra "h" falla y continúa.
- eventualmente llega a la h, retrocede 3, evalúa y continúa, luego hace match a la "h", luego a la "u" y así hasta emparejar toda la expresión.

Lookbehind de longitud variable

- La forma como funcionan los lookbehinds tiene una limitación.
- Se requiere que la expresión regular dentro del lookbehind tenga longitud finita y fija.
- Es decir que no podemos usar nada con cuantificadores.
- ¿Por qué? El autómata no sabría cuándo dejar de buscar.

¿Cuándo queremos lookbehind variable?

- Piense en un texto que tenga números de celular y valores monetarios.
- Estamos interesados en extraer únicamente los números de celular.
- Un lookbehind como este:

  "(? <!\$)[0-9]+"

  No puede funcionar.
- Quisiéramos este lookbehind

  "(? <!\$[0-9]*)[0-9]+"

  Pero no se puede.

Una solución ingeniosa

- Hay que entender la limitación en la forma como funciona el autómata finito.
- El secreto está en el uso de disjunciones y de los operadores **SKIP*** y **FAIL***
- Le vamos a permitir al programa que encuentre y haga match con lo que NO queremos, pero es una trampa. Le vamos a pedir que lo descarte.

$$NOT\_THIS|(THIS) \tag{1}$$

Una solución ingeniosa

- Con la expresión $REGEX1|(REGEX2)$ lo que queremos queda almacenado en el grupo de capturas número 1.
- Cómo la expresión que queremos está a la derecha del operador, siempre descarta primero lo que no queremos y deja sólo lo que queremos.
- Así funciona el autómata internamente.
- Sirve para excluir todo tipo de contextos.

Excluir varios tipos de contextos

$$TARZANIA|--TARZAN--|''TARZAN''|(TARZAN)$$

Implementación en PCRE

- Manejar los grupos de captura puede ser un poco engorroso para un programador que no le gusten.
- Digamos que uno está en R y quiere que su expresión le devuelva exactamente lo que quiere, sin complicaciones.
- En este caso nos valemos de los operadores (*SKIP)(*FAIL) que descartan el primer match, obteniendo un resultado análogo.

$$Not\_A(*SKIP)(*FAIL)|GetThis \tag{2}$$

Capturar TODO un contexto, un problema más difícil.

Recapitulemos sobre los cuantificadores

- Habíamos dicho que una expresión como "$a+$" capturaba " 1 o más ocurrencias de 'a"'
- ¿Qué significa "1 o más"?¿Cuántas va a coger?
- Los cuantificadores pueden ser avaros, egoístas, perezosos y generosos, como veremos más adelante.

Aplicación concreta

En un documento usted tiene números de cédula, montos monetarios, matrículas de registro, etc. Números que necesitan de su contexto completo para ser identificadas. Pero ese contexto puede estar detrás varias palabras. ¿Cómo se puede extraer?

Problema concreto

"El señor Francisco Barreras está **identificado** con la **cédula** de **ciudadanía C.C. 1019068721**. Esta **cédula** está registrada en la ciudad de Bogotá. El saldo de su cuenta de ahorros es de $1.000 pesos, pero en la cuenta figura con **identificación número 3106791236**."

- En este texto aparecen dos tipos de expresiones que nos interesan, unas palabras de contexto alrededor de las cédulas (necesitamos extraerlas para diferenciar esos números de otros)
- Otra distinta son los números como tal, queremos distinguirlos de otros números.

La solución es muy fácil Paco!

- Paco! ya sé cómo se hace!
- Algo como reg1 = "$Cedula|C.C.|Identificacion|Numero|Ciudadania$" va a coger las expresiones del primer tipo.
- Y algo como reg2= "$\$[0-9\.]+(*SKIP)(*FAIL)|[0-9\.]+$" va a coger la segunda!
- Simplemente usemos un *wildcard* para coger todo lo que está entre un reg1 y un reg2 y ya!

  "reg1.*reg2"
- **PACO:** Felicitaciones! pero esto no funciona :(

El kleene plus es avaro

- Quiere decir que el coge todas las instancias que pueda de la expresión anterior.
- En el caso del *wildcard* "." esto va a ser cualquier cosa, es decir, va a coger todo hasta el final del documento.
- ¿Cómo le digo que no sea tan avaro y que pare?
- sería fácil si reg2 fuera una expresión de un sólo caracter (¿Por qué?)

Volver al cuantificador perezoso

- En el mundo oscuro de las expresiones regulares avanzadas hay algo que se puede hacer.
- Si reemplazamos la expresión "(reg1).*" por la expresión "(reg1).*?" el cuantificador se vuelve perezoso.
- Es decir, captura todas las instancias posibles para que se pueda hacer el siguiente match. Osea, hasta que se choque con lo que sigue (no se lo engulle también).

GRACIAS