Regex Magic

Sumedhaa Kothari

12/03/2017

In my experience, Regular Expressions are a confusing entity. They are neither a coding language necessarily, nor a tool isolated to only R. Instead they are a bit magical in the sense that they perform a function across a variety of different languages.

The purpose of this post is not to dwell on what we have discussed regarding Regex in lecture. Rather, I want to share a more in depth history on Regex with you, perform a few interesting (harry potter related) examples, as well as discuss some of the run time limitations, and Regex's extensive capabilities (such as Wordles).

Regex is a pattern finder. As clarified on Wikipedia, it "defines a search pattern". It was derived from an idea for searching regular text expressions, by a mathematician named Stephen Kleene. It was not used in computer programming however, until 1968.

Regex allows the user to find patterns in strings and manipulate them, however they desire. There are an infinite amount of uses for this, such as replacing column names in a data frame with valid titles, or reformatting all of the data in a specific column into a consistent format. You can also search across large amounts of information (like webpages or census information) for certain types of data, such as phone numbers or license plates. This is because Regex allows you to do more than search only specific words like "cat" but also search for three numbers in a row, followed by a dash, followed by three more numbers, followed by another dash, followed by four numbers, as we saw in class. Seriously how cool is that?? In this case, you are the wizard and Regex is the wand.

Its time for a few examples.

```
# Load all required packages
library(stringr)
library(ggplot2)
```

Set up your data frame. I found a free repository of data on Harry Potter characters on Github.

```
# Save the web address as a variable
site <- 'https://raw.githubusercontent.com/efekarakus/potter-network/master/data/characters.csv'

#Download the contents of the file to a local directory, and then use the read.csv function to acquire the data, a
s it's stored as a CSV file online and on our machine.
download.file(site, destfile = 'HP.csv')

d <- read.csv('HP.csv')

summary(d)
```

```
##        id                          name
##  Min.   : 0   Alastor "Mad-Eye" Moody: 1
##  1st Qu.:16   Alberforth Dumbledore  : 1
##  Median :32   Albus Dumbledore       : 1
##  Mean   :32   Alice Longbottom       : 1
##  3rd Qu.:48   Aragog                 : 1
##  Max.   :64   Argus Filch            : 1
##               (Other)                :59
##                                                                          bio
##  Acromantula belonging to Rubeus Hagrid.                                   : 1
##  Albus' brother and owner of Hog's Head.                                   : 1
##  Best friend of Crabbe. Slytherin student and dies by falling into Fiendfyre.: 1
##  Best friend of James Potter and godfather of Harry.                       : 1
##  Betrays James and Lily Potter. Follower of Voldemort.                     : 1
##  Brother of Sirius. Used to be a Death Eater but defected.                 : 1
##  (Other)                                                                   :59
```

```
string <- list(d$name)
string
```

```
## [[1]]
##  [1] Regulus Arcturus Black       Sirius Black
##  [3] Lavender Brown               Cho Chang
##  [5] Vincent Crabbe Sr.           Vincent Crabbe
##  [7] Bartemius "Barty" Crouch Sr. Bartemius "Barty" Crouch Jr.
##  [9] Fleur Delacour               Cedric Diggory
## [11] Alberforth Dumbledore        Albus Dumbledore
## [13] Dudley Dursley               Petunia Dursley
## [15] Vernon Dursley               Argus Filch
## [17] Seamus Finnigan              Nicolas Flamel
## [19] Cornelius Fudge              Goyle Sr.
## [21] Gregory Goyle                Hermione Granger
## [23] Rubeus Hagrid                Igor Karkaroff
## [25] Viktor Krum                  Bellatrix Lestrange
## [27] Alice Longbottom             Frank Longbottom
## [29] Neville Longbottom           Luna Lovegood
## [31] Xenophilius Lovegood         Remus Lupin
## [33] Draco Malfoy                 Lucius Malfoy
## [35] Narcissa Malfoy              Olympe Maxime
## [37] Minerva McGonagall           Alastor "Mad-Eye" Moody
## [39] Peter Pettigrew              Harry Potter
## [41] James Potter                 Lily Potter
## [43] Quirinus Quirrell            Tom Riddle Sr.
## [45] Mary Riddle                  Lord Voldemort
## [47] Rita Skeeter                 Severus Snape
## [49] Nymphadora Tonks             Dolores Janes Umbridge
## [51] Arthur Weasley               Bill Weasley
## [53] Charlie Weasley              Fred Weasley
## [55] George Weasley               Ginny Weasley
## [57] Molly Weasley                Percy Weasley
## [59] Ron Weasley                  Dobby
## [61] Fluffy                       Hedwig
## [63] Moaning Myrtle               Aragog
## [65] Grawp
## 65 Levels: Alastor "Mad-Eye" Moody ... Xenophilius Lovegood
```

Now turn the data into a character vector, so it can be used with Regular Expresssions.

```
stext <- readLines('HP.csv')
```

Now, for a little magic- Find all of the Weasleys

```
grep("Weasley", stext)
```

```
##  [1] 10 23 27 52 53 54 55 56 57 58 59 60
```

Isn't that cool? You can see where Weasley is mentioned, whether it be part of a chacter's name or in a bio.

```
stext[grep("Weasley", stext)]
```

```
##  [1] "8,Fleur Delacour,Participated in the Triwizard tournament and married Bill Weasley."
##  [2] "21,Hermione Granger,One of Harry's best friend and marries Ron Weasley."
##  [3] "25,Bellatrix Lestrange,Death Eater who was killed by Molly Weasley."
##  [4] "50,Arthur Weasley,Father of the Weasleys and member of the Order of the Phoenix."
##  [5] "51,Bill Weasley,Oldest son of Arthur and Molly. Husband of Fleur. "
##  [6] "52,Charlie Weasley,Second son of Arthur and Molly. Works with dragons in Romania."
##  [7] "53,Fred Weasley,Identical twin with George and co-owner of Weasleys' Wizard Wheezes"
##  [8] "54,George Weasley,Identical twin with Fred and co-owner of Weasleys' Wizard Wheezes"
##  [9] "55,Ginny Weasley,Marries Harry Potter and only daughter of Molly and Arthur."
## [10] "56,Molly Weasley,Wife of Arthur and mother of the Weasleys. Kills Bellatrix."
## [11] "57,Percy Weasley,Third son of Arthur and Molly. He is a Gryffindor prefect."
## [12] "58,Ron Weasley,Harry's best friend. Marries Hermione."
```

Now say we only wanted to get the information on the Weasley characters… Ta da!

```
stext[grep("Weasley..", stext)]
```

```
## [1] "50,Arthur Weasley,Father of the Weasleys and member of the Order of the Phoenix."
## [2] "51,Bill Weasley,Oldest son of Arthur and Molly. Husband of Fleur. "
## [3] "52,Charlie Weasley,Second son of Arthur and Molly. Works with dragons in Romania."
## [4] "53,Fred Weasley,Identical twin with George and co-owner of Weasleys' Wizard Wheezes"
## [5] "54,George Weasley,Identical twin with Fred and co-owner of Weasleys' Wizard Wheezes"
## [6] "55,Ginny Weasley,Marries Harry Potter and only daughter of Molly and Arthur."
## [7] "56,Molly Weasley,Wife of Arthur and mother of the Weasleys. Kills Bellatrix."
## [8] "57,Percy Weasley,Third son of Arthur and Molly. He is a Gryffindor prefect."
## [9] "58,Ron Weasley,Harry's best friend. Marries Hermione."
```

Now, while most of this manipulation is quite basic, I provide it only to highlight how transfomative Regex was in its origin. It really has the capability to do many manipluations on strings, however that is not the main focus of this post. Rather I want to discuss a few other interesting Regex related facts I read about in my exploration of this post.

Searching for liscence plates, or phone numbers, by only specifying numbers is called using wild cards. While wildcards are a useful component of Regex, improper use of wildcards greatly affects runtime of certain expressions. In her article, Regexes: The Bad, the Better, and the Best, Liz Bennett describes how a Regex expression written three different ways can alter the run time from 10,000 milliseconds for a bad expression vs. 420 milliseconds for a good one. This is because Regex is a tool that searches across text, which worst case can be an extremely timely task if it is searching for every combination of number or character, with very few parameters. So what distinguishes a good expression from a bad one? The main aspect is the amount of specificity within an expression using wildcards. I.e. the bad Regex expression used /* and did not specify a limit to how many numbers could follow. The good expression placed limits on how many numbers should be allowed to be searched for and when numbers should be followed by characters and vice versa.

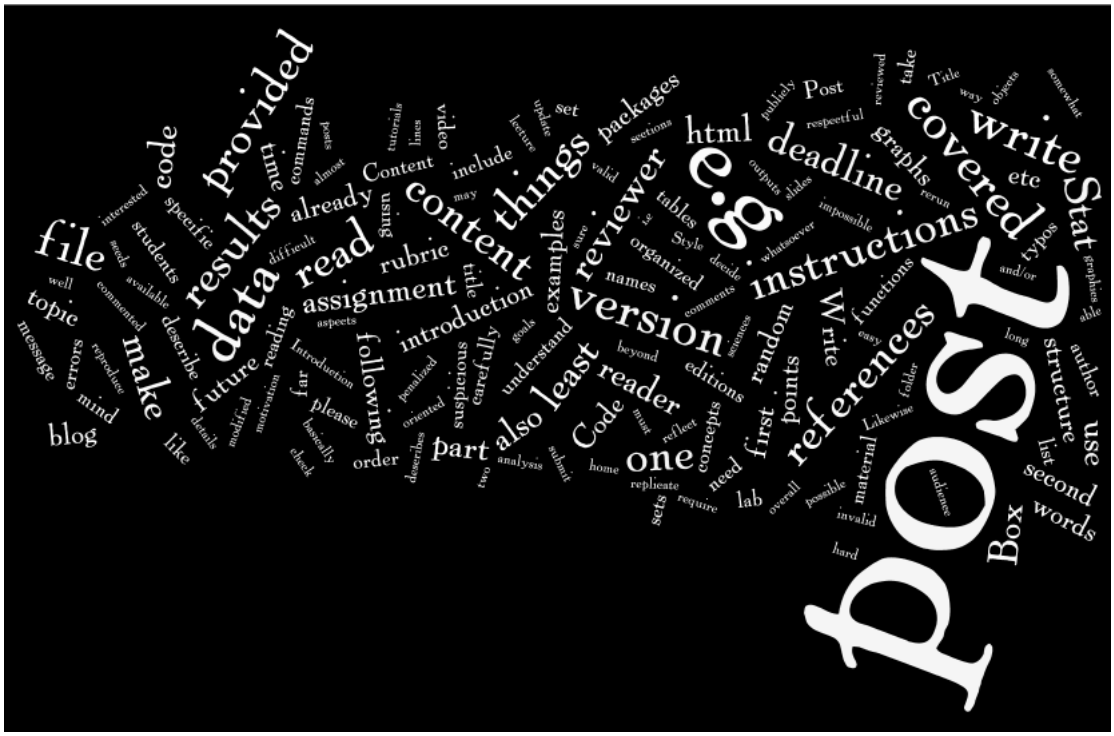
Something that is really cool about Regex and its capabilites is the extension into realms such as wordle! As described in Wordle, by Jonathan Feinberg, wordles are often made with Regex, using java! Here is an example of how its done:

Example 3-1. *How to recognize "words"*

```
private static final String LETTER = "[@+\\p{javaLetter}\\p{javaDigit}]";
private static final String JOINER = "[-.:/''\\p{M}\\u2032\\u00A0\\u200C\\u200D~]";
/*
A word is:
    one or more "letters" followed by
    zero or more sections of
        one or more "joiners" followed by one or more "letters"
*/
private static final Pattern WORD =
    Pattern.compile(LETTER + "+(" + JOINER + "+" + LETTER + "+)*");
```

* See *http://github.com/vcl/cue.language*.
† For an illuminating demonstration of this craft, see Peter Norvig's chapter on natural-language processing in the sister O'Reilly book *Beautiful Data*.

And here is a wordle of the guidlines for ~Post 02~!!



References:

- https://www.r-bloggers.com/regular-expressions-in-r-vs-rstudio/
- https://en.wikipedia.org/wiki/Regular_expression
- https://users.soe.ucsc.edu/~pang/261/f15/misc/wordle.pdf
- http://cs.lmu.edu/~ray/notes/regex/
- https://www.loggly.com/blog/regexes-the-bad-better-best/
- https://code.tutsplus.com/tutorials/8-regular-expressions-you-should-know--net-6149
- http://www.gastonsanchez.com/r4strings/regex1.html