Introduction to Regular Expressions

Cedric Arisdakessian

2022-09-02

A simple example

Your team compiled a metadata file for your project. One of the samples attribute is the location with the name of the island. However, each person has its own way to spell Hawaii:

- hawaii
- Hawaii
- Hawai’i
- Hawai‘i

When you load your metadata into R, your island column has much more levels than it should have (25 instead of 5). It’s because R doesn’t know the different spellings correspond to the same island.

How would you fix this issue?

1. You don't have that many samples, you can just fix it manually.
2. You decide to hire an intern to fix it.
3. You look for all possible patterns in excel and replace them.
4. You have too many samples and you're not sure of all the spellings. You decide to learn regular expressions.

What are regular expressions

It's a standardized language to define patterns of text, and it is commonly used to:

- Extract patterns
- Find/Replace patterns

Many tools use regex:

- Computer science languages (R, python, ... actually most of them)
- Linux command-line tools (grep, sed)
- Probably even excel in some way

By the end of this lecture, you should be able to decipher patterns like:

```
[A-Za-z0-9_\-]+@[A-Za-z0-9_\-]+\.[a-z]{2,}$
```

Throughout the lecture, we are going to use R to see if a regex matches a text. We can do that with the `grepl(pattern, text)` function.

Before we start

- The regex language uses common characters to express patterns of text (ex: `-`, `?`, `.`)
- What if you need to look for one of them in a text?
  ⇒ By default, it will be assumed that you mean the regex interpretation.
  ⇒ If you mean to match the actual symbol, you need to escape it (e.g. `\.` for a literal dot). In R, you will need 2 backslashes.

Dot symbol

The regex syntax is a set of symbols with a specific meaning:

- "." refers to any character.

```
grepl(".", "a")
```

```
## [1] TRUE
```

```
grepl("\\.", "a")
```

```
## [1] FALSE
```

```
grepl(".", "$")
```

```
## [1] TRUE
```

Matching specific location in the text

- "^" matches the beginning of the text.
- "$" is similar to "^" but matches the end

```
# Look for letter "c" at the beginning of the text
grepl("^c", c("cedric", "michaela"))
```

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

```
# Look for letter "R" at the end of the text
grepl(".R$", c("main.R", "Rstudio.app"))
```

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

```
# bonus: what might be an issue with the previous example?
```

Combine regex

AND

If two regex follow each other (and are inside square brackets), the patterns combine:

```
grepl("a.a", "aba")
```

```
## [1] TRUE
```

```
grepl("a.a", "abba")
```

```
## [1] FALSE
```

OR

`"x|y"` will match text if it matches either x or y

```
grepl("a|b", "a")
```

```
## [1] TRUE
```

```
grepl("(a|b)c", "ac")
```

```
## [1] TRUE
```

Multiple choices and ranges

- The bracket "[xyz]" notation is similar and usually preferred for more than 2 choices
- The "[x-y]" notation refers to ranges (works for letters as well)

```
grepl("[ct]sv$", ".csv")
```

```
## [1] TRUE
```

```
grepl("[A-Z]", c("A", "a"))
```

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

```
grepl("[A-Za-z0-9 ]", c("A", "a", "3", " "))
```

```
## [1] TRUE TRUE TRUE TRUE
```

Repetition of symbols

- "*" refers to 0 or more repetition of the previous pattern. Note that it's a bit different that its meaning as a glob pattern.
- "+" refers to 1 or more repetition of the previous pattern

```
grepl("a*", c("a", "", "aa"))
```

```
## [1] TRUE TRUE TRUE
```

```
grepl("a+", c("a", "", "aa"))
```

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

```
grepl("aa*", c("a", "", "aa"))
```

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

You can pick the number of repetitions with the `"{n, m}"` syntax

```
grepl("ba{2,3}b", c("baab", "baaaab"))
```

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

```
grepl("a{2,}", c("a", "aa", "aaaa"))
```

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

Warning

```
grepl("a{,2}", "aaaaa")
```

```
## [1] TRUE
```

```
grepl("^a{,2}$", "aaaaa")
```

```
## [1] FALSE
```

Negation

When at the beginning of square brackets, the "^" symbol negates a set of characters:

```
grepl("b[^bB]b", c("bbb", "bBb", "bab"))
```

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

Summary

- `"."`: any character.
- `x|y`: x OR y
- `[xyz]`: x OR y OR z
- `[x-y]`: any character in range (numbers or letters)
- `[^xyz]`: anything but x, y or z
- `"^"`: matches the beginning of the text.
- `"$"`: matches the end of the text
- `"*"`: 0+ repetitions
- `"+"`: 1+ repetitions
- `{n,m}`: between n and m repetitions

More regex

There's actually more to the regex syntax that is presented here. Once you're familiar with the ones presented here, it might be interesting looking into:

- capture groups (select one or multiple groups and reorganize them)
- positive/negative lookahead (need to understand capture groups first)

Using regex in R

- `grep`, `grepl` : Find in string
- `gsub` : string substitution

```
# gsub(pattern, replacement, text)
gsub("a", "b", "aaa")
```

```
## [1] "bbb"
```

```
gsub("a", "b", c("aaa", "cac"))
```

```
## [1] "bbb" "cbc"
```

Using regex in linux: grep

`grep [PATTERN] [FILE]` find the lines matching a given pattern in a file. (many additional options)

```
# Count the number of sequences in a fasta file
$ grep -c "^>" sequences.fasta

# Count the number of sequences in a fastq file
$ grep -c "^@" reads.fastq # roughly
$ grep -c "^@M01" reads.fastq # a bit better

# Retrieve the entries labelled as S.aureus in the header of a f
# (needs to be 2-line formatted)
$ grep -A1 "^>.*S\.aureus.*"
# or
$ grep -A1 "S\.aureus"
```

Using regex in linux: sed

Very complex command. But basic functionalities are more manageable:

- `sed 's/[pattern]/[replacement]/'` (replace the first occurrence of `[pattern]` with `[replacement]` in each line)
- `sed 's/[pattern]/[replacement]/g'` (replace any occurrence of [pattern] with [replacement] in each line)
- `sed '[line_pattern]/s/[pattern]/[replacement]/g'` (match only lines with pattern)
- `sed '[line_pattern]/!s/[pattern]/[replacement]/g'` (negate line pattern)

And much more.

```
# Replace excel "#VALUE" in csv file (inplace)
$ sed -i 's/#VALUE/NaN/g' metadata.csv
# More complex: Remove all N nucleotides in a fasta file
$ sed '^>/!s/N//g' sequences.fasta > sequences_N-removed.fasta
```