Regular expressions

Max Turgeon

SCI 2000–Introduction to Data Science

Lecture Objectives

- Understand the definition of regular expressions
- Recognize and use the different metacharacters
- Use regular expressions to filter and edit text data

Motivation

Regular expression–Definition

Applications of regexes

Example i

```
library(stringr)
```

Anchors

- **Anchors** are special characters (i.e. metacharacters) that can be used to specify *where* we want to find a match.
- There are two main anchors:
  - `^pattern` will match any string that *starts* with `pattern`
  - `pattern$` will match any string that *ends* with `pattern`
- You can combine them:
  - `^pattern$` will *only* match the string `pattern`
- If you want to match on a metacharacter (e.g. `$`), you need to escape it (see example below).

Example i

```
# This doesn't work...
str_detect(c("$15.99", "$3.75", "1.99$"),
           pattern = "^$")
```

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

```
# But this does!
str_detect(c("$15.99", "$3.75", "1.99$"),
           pattern = "^\\$")
```

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

Example ii

```
# Matching on prices that start or end with dollar sign
# Use logical operator
str_detect(c("$15.99", "$3.75", "1.99$"),
           pattern = "^\\$") |
  str_detect(c("$15.99", "$3.75", "1.99$"),
           pattern = "\\$$")
```

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

Quantifiers

- **Quantifiers** are ways to specify how many times a certain pattern should appear.
    - At least once? Exactly three times?
- There are four important metacharacters to remember.
    - `.` will match any single character, except a new line.
    - `?` will match the item on its left at most once.
    - `*` will match the item on its left zero or more times.
    - `+` will match the item on its left once or more times.
- Key distinction between `*` and `+`: the latter requires at least one match.

Example i

Quantifiers cont'd

- You can also control the number of matches more precisely.
  - `{n}` will match the item on its left exactly n times.
  - `{n,}` will match the item on its left at least n times.
  - `{n,m}` will match the item on its left at least n times, but no more than m times.

Exercise

Find a regular expression that matches string ending with an ellipsis (i.e. three dots).

Solution

```
str_detect(c("string.", "string..", "string..."),
           pattern = "\\.{3}$")
```

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

```
# Be careful: a string with 4 dots will also match
str_detect("string....", pattern = "\\.{3}$")
```

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

Character classes i

- When discussing quantifiers, I used “item on the left” instead of “character on the left”.
- This was intentional: you can create **character classes** using square brackets.
    - E.g. `p[ao]rt` will match both `part` and `port`.
- Character classes can also be created using *sequences*.
    - E.g. `[a-z]` will match all lower case letters; `[a-zA-Z]` will match all lower and upper case letters; `[0-9]` will match all ten digits.

Character classes ii

- There are also built-in character classes:
  - `\\d` matches any digit character (equivalent to `[0-9]`)
  - `\\s` matches any space character (including tabs, new lines, etc.)
  - `\\w` matches any word character (equivalent to `[A-Za-z0-9_]`)
  - `\\b` matches word boundaries
- Finally, you can negate character classes to get non-matches.
  - `p[^ao]rt` matches `purt` and `pert`
  - The negation of `\\d`, `\\s`, `\\w`, `\\b` are `\\D`, `\\S`, `\\W`, `\\B` respectively.

Example i

```
# Split a sentence into words
str_split("The fox ate a berry.", "\\b")
```

```
## [[1]]
## [1] "" "The" " " "fox" " " "ate" " " "a" " "
## [10] "berry" "."
```

```
str_split("The fox ate a berry.", "\\s")
```

```
## [[1]]
## [1] "The" "fox" "ate" "a" "berry."
```

Example ii

```
# Trim white space
str_replace_all("Is  this     enough?",
                pattern = "\\s+",
                replacement = " ")
```

```
## [1] "Is this enough?"
```

Exercise

Find a regular expression that matches white space at the beginning and the end of a string.

Solution

```
str_replace_all(" Is this enough?   ",
                pattern = "(^\\s+|\\s+$)",
                replacement = "")
```

```
## [1] "Is this enough?"
```

Summary