Regular Expressions

Hauke Licht
University of Cologne

April 24, 2024

Intro

Regular expressions

- regular expressions = `regex`
- powerful tools for working with text data
- allow to describe and identify patterns in text through **abstraction**
- makes pattern searching, replacing, and parsing much easier

What are regex and why do we need them

- **Definition**: Regular expressions (short **regex**) are specialized symbols and operators that define a search pattern.
- **Utility**: Useful for string searching and manipulation tasks in programming.
- **Applications**: Validating text input, searching in documents, and transforming text data.

Setup

```
1 library(stringr)
```

Basics

The Placeholder .

The dot . matches any single character except newline characters.

Example

the regex pattern “a.c”, . will match any pattern that start with ‘a’, ends with ‘c’, and has an abritrary character in between

```
text <- c("abc", "adc", "a?c", "a_c", "a̸c", "ac")
grepl("a.c", text)
```

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

Special Symbols

Special symbols in regex help to match specific types of characters like digits, words, or spaces.

Word Character `\\w`

- `\\w` represents any word character which includes letters, digits, and the underscore

Example

```
text <- "Hello world 123!"
str_extract_all(text, "\\w+")
```

```
[[1]]
[1] "Hello" "world" "123"
```

Special Symbols

Special symbols in regex help to match specific types of characters like digits, words, or spaces.

Digit \\d

\\d represents any digit from 0 to 9

Example

```
text <- "In 2023, we 1½ million Euros."
str_extract_all(text, "\\d+")
```

```
[[1]]
[1] "2023" "1"
```

Special Symbols

Special symbols in regex help to match specific types of characters like digits, words, or spaces.

Space `\\s` and Variants

`\\s` is Geared towards identifying different types of spaces in text.

Example

```
text <- "In 2023, we 1½ million     Euros."
str_extract_all(text, "\\s+")
```

```
[[1]]
[1] " "     " "     " "     " "     "     "
```

Special Symbols

Special symbols in regex help to match specific types of characters like digits, words, or spaces.

Horizontal Space \\h

\\h represents**: Any horizontal whitespace like regular spaces or tab characters.

Example

```
text <- "Some text\t with a tab."
str_extract_all(text, "\\h+")
```

```
[[1]]
[1] " "   "\t " " "   " "
```

Special Symbols

Special symbols in regex help to match specific types of characters like digits, words, or spaces.

Vertical Space `\\v`

`\\v` represents any vertical white spaces like line breaks.

Example

```
1  text <- "Line one\nLine two\r\n"
2  str_extract_all(text, "\\v+")
```

```
[[1]]
[1] "\n"   "\r\n"
```

Repetition Quantifiers

Repetition quantifiers specify how many times elements can repeat in sequence

Zero times or more: *

Matches 0 or more repetitions.

Example

`"lo*"` matches any ‘l’ followed by zero or more ’o’s.

```
text <- "hellooo"
str_extract_all(text, "lo*")
```

```
[[1]]
[1] "l"    "looo"
```

Repetition Quantifiers

Repetition quantifiers specify how many times elements can repeat in sequence

Zero or one time: ?

Example

`"lo?"` matches any ‘l’ followed by none or one ‘o’.

```
1  text <- "hellooo"
2  str_extract_all(text, "lo?")
```

```
[[1]]
[1] "l"  "lo"
```

Repetition Quantifiers

Repetition quantifiers specify how many times elements can repeat in sequence

One time or more: +

`+` matches one or more repetitions.

Example

`"lo+"` matches ‘l’ followed by one or more ’o’s.

```
text <- "hellooo"
str_extract_all(text, "lo+")
```

```
[[1]]
[1] "looo"
```

Repetition Quantifiers

Repetition quantifiers specify how many times elements can repeat in sequence

Between min and max times: `{min,max}`

`{min,max}` matches between a minimum and maximum number of occurrences.

Example

`"lo{1,2}`" matches 'l' folloewd by between 1 and 2 'o's

```
text <- "hellooo"
str_extract_all(text, "lo{1,2}")
```

```
[[1]]
[1] "loo"
```

Note: `{0,1}` is equivalent to using `?`

Defining Character Sets and Ranges with [ ]

- [ ] Allows you to match specific sets or ranges of characters
- just list all characters or regex operators that should be matched

Example

```r
text <- "Match1234!"
str_extract_all(text, "[M012!]+")
```

```
[[1]]
[1] "M"   "12" "!"
```

Note: the set can also include special regex operators

```r
text <- "Match1234!"
str_extract_all(text, "[\\w\\d]+")
```

```
[[1]]
[1] "Match1234"
```

Defining Character Sets and Ranges with `[ ]`

- `[]` Allows you to match specific sets or ranges of characters
- just list all characters
- or use `-` to define ranges

Examples

- **Numbers**: `[0-9]` matches any digit.
- **Lower case letters**: `[a-z]` matches any lower case letter.
- **Upper case letters**: `[A-Z]` matches any upper case letter.

```
1  text <- "Match1234!"
2  str_extract_all(text, "[a-zA-Z0-9]+")
```

```
[[1]]
[1] "Match1234"
```

Note: Because `-` has the function of defining ranges, you need to include it last in a set when you want to match it literally

```
1  text <- "Match-1234!"
2  str_extract_all(text, "[a-z-]+")
```

Pre-defined Character Sets

`[]` Allows you to match specific sets or ranges of characters

Special sets

- **Alpha characters**: `[[:alpha:]]` matches all alphabetic characters.
- **Alphanumeric**: `[[:alnum:]]` matches all alphanumeric (digits and letters).

```r
text <- "Run3Forest, Run!"
str_extract_all(text, "[[:alpha:]]+")
```

```
[[1]]
[1] "Run"    "Forest" "Run"
```

```r
str_extract_all(text, "[[:alnum:]]+")
```

```
[[1]]
[1] "Run3Forest" "Run"
```

Note: Special sets can be used inside more complex sets

```r
text <- "Run3Forest, Run!"
str_extract_all(text, "[[:alpha:] ,]+")
```

```
[[1]]
[1] "Run"         "Forest, Run"
```

Negating Character Sets and Ranges inside [ ]

- [ ] Allows you to match specific sets or ranges of characters
- if you put an ^ at the first position inside the brackets, it negates the set

Examples

```r
text <- "Match1234!"
str_extract_all(text, "[^a-zA-Z0-9]+")
```

```
[[1]]
[1] "!"
```

```r
text <- "Run3Forest, Run!"
str_extract_all(text, "[^[:alpha:]]+")
```

```
[[1]]
[1] "3"  ", " "!"
```

```r
str_extract_all(text, "[^[:alnum:]]+")
```

```
[[1]]
[1] ", " "!"
```

Anchors: ^ and $

- **Function**: Anchor symbols to match the beginning ^ and end $ of a line.

Examples

- "^M" matches lines starting with 'M'
- "ing$" matches lines ending with 'ing'

```
1 text <- c("Morning", "Evening")
2 grepl("^M", text)
```

```
[1]  TRUE FALSE
```

```
1 grepl("ing$", text)
```

```
[1] TRUE TRUE
```

Notes:

- ^ and $ can be used together to match whole lines
- remember from the previous slides that ^ used inside [...] has a different function (it negates the set)

Grouping with (. . .)

(. . .) enables logical grouping of part of a pattern and can be used together with quantifiers

Example

```r
text <- c("banana", "ananas", "Shalalalala")
str_extract_all(text, "(na)+")
```

```
[[1]]
[1] "nana"

[[2]]
[1] "nana"

[[3]]
character(0)
```

Important: Because (and) are used as special symbols, they need to be escaped with \\ when used as literal characters.

```r
text <- "+49 (0) 176 1234 56 78"
str_extract_all(text, "\\(0\\).+")
```

```
[[1]]
[1] "(0) 176 1234 56 78"
```

Using | for Combining Patterns (“Alternation”)

- **Function**: The | character (“pipe”) acts as an OR operator.
- **Example**: `"cat|dog"` matches ‘cat’ or ‘dog’.

```
text <- "cats and dogs"
str_extract_all(text, "(cat|dog)")
```

```
[[1]]
[1] "cat" "dog"
```

Exercises

Exercises

Read the sample of the ParlSpeech2 *UK Hose of Commons* corpus I have created

```
library(readr)
fp <- file.path("data", "datasets", "parlspeech2_gbr_sample.tsv")
df <- read_tsv(fp)
```

1. What is the share of speeches that starts with the words “The” or “My”.
2. “hon.” who?
    1. Extract all occurrences of the pattern “hon.” followed by a white space and a title-case word
    2. Count the number of times the words after “hon.” occur

Using regex for data reshaping

the `tidyr` pattern provides helpful functions for reshaping data using regex.

- **`separate()`**: splits a column into multiple columns based on a regex pattern

Examples

```r
text <- c("John was born in 1980", "Mike was born in 1990")
df <- data.frame(text)
print(df)
```

```
                 text
1 John was born in 1980
2 Mike was born in 1990
```

```r
tidyr::separate(df, text, into = c("name", "year"), sep = " was born in ")
```

```
  name year
1 John 1980
2 Mike 1990
```

Using regex for data reshaping

the `tidyr` pattern provides helpful functions for reshaping data using regex.

- `separate()`: splits a column into multiple columns based on a regex pattern
- **`seperate_rows()`**: splits a column into multiple rows based on a regex pattern

Examples

```r
# example for use of separate_rows()
df <- data.frame(
  group = c("A", "B"),
  text = c("John, Mike", "Alice, Bob, and Luis")
)
print(df)
```

```
  group                 text
1     A           John, Mike
2     B Alice, Bob, and Luis
```

```r
tidyr::separate_rows(df, text, sep = ", (and )?")
```

```
# A tibble: 5 × 2
  group text
  <chr> <chr>
1 A     John
2 A     Mike
3 B     Alice
4 B     Bob
5 B     Luis
```

Exercises (*continued*)

MPs in the *House of Commons* sometimes address other MPs by referring to their constituency.

The pattern

- starts with “Member” or “Members”
- might be followed by “of Parliament”
- next comes “for” and the constituency name, which consists of one ore more title-case words (might be connected with a “-”) that might be separated by a comma or an “and”
- the constituency name might be followed by the MP’s name in parentheses
- if several members are mentioned, individual constituencies are separated by “and” and each starts with “for”

Examples

```
"Member for Stratford", "Member for Halifax (Mrs Riordan)", "Member for South-West Norfolk (Mrs. Shephard)",
"Member for Hamilton, South (Mr. Tynan)", "Member for Edinburgh, East and Musselburgh (Dr. Strang)",
"Members for South Derbyshire (Mr. Todd) and for North Durham"
```

1. Extract all occurrences of this pattern.
2. Which are the 20 most (least) frequently mentioned constituencies?
3. Which are the 20 most (least) frequently mentioned MPs?