R programming, a gentle introduction

M1 IREF, M1 ROAD

Laurent Bergé

B_xSE, University of Bordeaux

Fall 2022

Text manipulation

Character strings in R

- nowadays, many applications using text data (NLP & ML)
- pretty easy to deal with text in R (and usually efficient)
- we will see only the most basic (and usually most useful) text manipulation in R

Concatenate strings: Paste

To concatenate several character strings (henceforth CS), use `paste()`:

```
paste("hi", "everyone")
#> [1] "hi everyone"
godNames = c("Zeus", "Aphrodite")
paste("Hello holy", godNames) # returns a vector
#> [1] "Hello holy Zeus"      "Hello holy Aphrodite"
```

- This function takes in several vectors and return one vector.

```
# What's the result of:
paste(c("iphone", "Samsung"), 1:6)
```

```
#> [1] "iphone 1"  "Samsung 2" "iphone 3"  "Samsung 4" "iphone 5"  "Samsung 6"
```

- To understand the behavior: remember **recycling**!

Paste

- This function can return either scalar or vectors. It has two main arguments:

1. `sep`, which is the separator between two CS (default is `" "`)
2. `collapse`, if provided, it will glue a vector of CS with the value of `collapse`

```
paste("Hello holy", godNames, collapse = " and ") # 1 CS only
#> [1] "Hello holy Zeus and Hello holy Aphrodite"
paste("Hello holy", godNames, sep = "....") # vector of length 2
#> [1] "Hello holy....Zeus"      "Hello holy....Aphrodite"
paste("Hello holy", godNames, sep = "....", collapse = " and ")
#> [1] "Hello holy....Zeus and Hello holy....Aphrodite"
```

The behavior of collapse is:

1. `charvec_tmp = paste("Hello holy", godNames, sep = ".....")`
2. `paste(charvec_tmp, collapse = " and ")`

Paste's friend `paste0`

By default the function `paste` concatenates with a space between the character elements:

```
paste("20", "22")
#> [1] "20 22"
```

Use `paste0` to concatenate with the empty string:

```
paste0("20", "22")
#> [1] "2022"
```

I avoids adding the argument `sep = ""` in `paste`.

Paste: exercise

Let `df = iris` a copy of the `iris` data.

Create a unique ID for `df` observations.

The character ID should be of the form:

[Species name]_[order of appearance].

```
# something that may be useful
table(iris$Species) # frequencies
#>
#>     setosa versicolor  virginica
#>         50         50         50
```

Formatting character vectors

```
"Laurent" == "laurent"
#> [1] FALSE
"bergé" == "berge"
#> [1] FALSE
"Laurent, Bergé" == "Laurent Bergé"
#> [1] FALSE
```

- as you can see, although the values convey the same information, they are treated as different.
- when dealing with text data, you first need to format them for meaningful comparisons.

Converting to ASCII

To convert to ASCII, easiest way is to use `iconv()`:

```
iconv("Laurent Bergé, €™", to = "ASCII")
#> [1] NA
iconv("Laurent Bergé, €™", to = "ASCII//IGNORE")
#> [1] "Laurent Berg, "
iconv("Laurent Bergé, €™", to = "ASCII//TRANSLIT")
#> [1] "Laurent Berge, ?T"
```

- argument `to` defines the bahavior of `iconv`:

1. "ASCII" defines the encoding target. By default, if non ACII character is encounters: full value is `NA`.
2. "IGNORE": if a non ASCII is met, it is deleted.
3. "TRANSLIT": if a non ASCII is met, it is replaced with an "equivalent" letter -- or a question mark if no equivalent is found.

Formatting: lower/upper

```
foxDog = "The Brown Fox Jumps Over The Lazy Dog"
tolower(foxDog)
#> [1] "the brown fox jumps over the lazy dog"
toupper(foxDog)
#> [1] "THE BROWN FOX JUMPS OVER THE LAZY DOG"
```

Extracting substrings

```
# to extract a subset of a CS:
substr(foxDog, start = 1, stop = 13)
#> [1] "The Brown Fox"
substr(foxDog, 26, nchar(foxDog))
#> [1] "The Lazy Dog"
```

```
# You can apply it directly to vectors
substr(rep(foxDog, 2), c(1, 26), c(13, nchar(foxDog)))
#> [1] "The Brown Fox" "The Lazy Dog"
```

Splitting

To split a CS, use `strsplit()`:

```
strsplit(foxDog, split = "Jumps Over")
#> [[1]]
#> [1] "The Brown Fox " " The Lazy Dog"
```

What do you notice?

1. The splitting character disappeared
2. It returns a list! $\Rightarrow$ What's the logic?

Splitting

```
# It can be applied to vectors:
text = c("Rumble thy bellyful!", "Spit, fire!", "Spout, rain!",
         "Nor rain, wind, thunder, fire are my daughters.")
strsplit(text, split = " ")
#> [[1]]
#> [1] "Rumble"    "thy"       "bellyful!"
#>
#> [[2]]
#> [1] "Spit," "fire!"
#>
#> [[3]]
#> [1] "Spout," "rain!"
#>
#> [[4]]
#> [1] "Nor"        "rain,"      "wind,"      "thunder,"   "fire"
#> [6] "are"        "my"         "daughters."
```

- you can apply `strsplit()` to vectors. Since the number of elements can be varying, returning a list is natural
- don't forget brackets, `strsplit(text, split)[[1]]`, for single CS

Splitting: Exercise I

Let's look at this corpus:

```
textvec = c("The Brown Fox Jumps Over The Lazy Dog",
            "Nor rain, wind, thunder, fire are my daughters.",
            "When my information changes, I alter my conclusions.")
textvec_split = strsplit(textvec, " ")
```

- recreate a character vector whose elements are the first 4 words of each text.

Splitting: Exercise II

The file `stopwords_en.RData` contains English stopwords (common words usually relating no specific meaning).★

The operator `x %in% s` asks whether the elements of a vector `x` belong to the set `s`.

```
5 %in% 1:5
#> [1] TRUE
"bonjour" %in% c("bonjour", "les", "gens")
#> [1] TRUE
c("bonjour", "au revoir") %in% c("bonjour", "les", "gens")
#> [1]  TRUE FALSE
```

Use `%in%` to recreate the following vector of text without stopwords:

```
textvec = c("The Brown Fox Jumps Over The Lazy Dog",
            "Nor rain, wind, thunder, fire are my daughters.",
            "When my information changes, I alter my conclusions.")
```

★: Use the function `load` to open it.

Replacing text within text

Say you have the following sentence:

The king infringes the law on playing curling.

Task

You want to stem the sentence, i.e. taking off the "ing" to keep only the root of the words.

Solution?

The function `gsub()` takes in a character string and replaces a string pattern with another string.

```
# the arguments are the original order of gsub
gsub(pattern = "jour", replacement = " soir", x = "Bonjour")
#> [1] "Bon soir"
```

Trying gsub

So let's stem the sentence with `gsub`.

Let's suppress all the `"ing"`:

```
kingText = "The king infringes the law on playing curling."
gsub(pattern = "ing", replacement = "", x = kingText)
#> [1] "The k infres the law on play curl."
```

Hmm, this was too strong, `infringe` became `infre`, let's give it another shot:

```
# a space is added after "ing"
gsub("ing ", " ", kingText)
#> [1] "The k infringes the law on play curling."
```

That's better. But unfortunately new problems pop:

1. `curling` now is not treated
2. `king` became `k` and its meaning is completely lost

gsub and regular expressions

We can easily deal with the two issues with regular expressions!

```
gsub("([[:alpha:]]{3,})ing\\b", "\\1", kingText)
#> [1] "The king infringes the law on play curl."
```

- Regular expressions are *extremely powerful* tools to deal with text data.
- Regular expressions are a language *per se* which takes time to master, but it's worth it.
- Regular expressions can be used in many (all?) programming languages!

Regular expressions

In this course I'll detail only a few important features.

For more detailed information, look at `?regexp` or the many regular expression tutorials existing.

regex: Special flags

In a regex, **two** backslashes, `\\`, are used for special characters.

`\\b` means the end of a word, a word consisting of a succession of letters or digits.

```
gsub("ing\\b", "", kingText) # now works for "curling."
#> [1] "The k infringes the law on play curl."
```

regex: The square brackets

The special argument `[]` means: any character that matches what's inside the brackets.

```
gsub("[aeiouy]", "_", kingText)
#> [1] "Th_ k_ng _nfr_ng_s th_ l_w _n pl___ng c_rl_ng."
```

- Any vowel is replaced with "_".
- The special argument `[:alpha:]` works *only inside brackets* and means all the alphabet:
  `[[:alpha:]]` is equiv. to
  `[abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ]`

regex: Predefined elements

```
gsub("[[:alpha:]]", "_", kingText)
#> [1] "___ ____ _________ ___ ___ __ _______ _______."
```

- Only non-letters are not replaced (the space and the point).
- Other examples are `[:digit:]` and `[:punct:]`.
- You can put anything you want in the brackets argument: e.g. `[[:punct:]123 ]` to match any punctuation, space or digits from 1 to 3.

regex: Multiple matching

When you want a pattern to be matched several times:

1. `{a, b}` means: previous pattern appears at least `a` times and at most `b` times
2. `+` means: previous pattern appears at least once (equiv. `{1, }`)
3. `*` means: previous pattern appears 0 or more times (equiv. `{0, }`)★

Question

What does the following do?

```
gsub("\\b[[:alpha:]]{1,3}\\b", "_", kingText)
```

```
gsub("\\b[[:alpha:]]{1,3}\\b", "_", kingText)
#> [1] "_ king infringes _ _ _ playing curling."
```

★: Yes, this is useful.

regex: Anything

- the spacial value "`.`" means "anything"

Say you want to delete everything after the word `king`:

```
gsub("king.+", "king", kingText)
#> [1] "The king"
```

regex: Escaping and conditions

- as you've seen some characters have a special meaning in regular expressions, so if you want to match them, you have to escape them with `\\`
- use "`|`" to mean OR

```
text = "[my.text.in.brakets]"
gsub("[", "", text) # error
#> Warning in gsub("[", "", text): TRE pattern compilation error 'M
#> Error in gsub("[", "", text): invalid regular expression '[', re

gsub("\\[", "", text) # OK
#> [1] "my.text.in.brakets]"

gsub("\\[|\\.|\\]", " ", text) # pipe means "or"
#> [1] " my text in brakets "
```

regex: Dynamic replacements

In the replacement, the special argument `\\1` means the first element that is in between parentheses.

Question

What does that do?

```
text = "abc123 x22 work 32"
gsub("([[:alpha:]]+)([[:digit:]]+)", "\\2\\1", text)
```

```
text = "abc123 x22 work 32"
gsub("([[:alpha:]]+)([[:digit:]]+)", "\\2\\1", text)
#> [1] "123abc 22x work 32"
```

regex: Summing up

With all our new knowledge, you now understand how this works:

```
gsub("([[:alpha:]]{3,})ing\\b", "\\1", kingText)
#> [1] "The king infringes the law on play curl."
```

regex: Exercise

Create the following regular expressions:

1. to delete words finishing with a `s`
2. to drop all terminal `s` when a word is at least 3 letters long (without the `s`).

- Test on:

```
text = "These guys like rhymes."
```

Text in R: Random tips and beyond

- To find out which CS matches the regex, `grepl()`:

```
text = c("hello", "folks", "goodbye")
grepl("e", text)
#> [1]  TRUE FALSE  TRUE
```

- to improve the speed for large vectors: use argument `perl = TRUE`
- other resources:
  - nice cheat sheet on regular expressions: from Rstudio
  - the package `stringr` provides user-friendly version of base R functions
  - R's task view on Natural Language Processing for an overview of many tools regarding NLP