Text Mining and Sentiment Analysis

Prof. Annamaria Bianchi
A.Y. 2024/2025

Lecture 4

25 February 2025

UNIVERSITÀ DEGLI STUDI DI BERGAMO | Dipartimento di Scienze Economiche

Outline

Operator precedence and parenthesis

Determine which strings match a pattern

Keep strings matching a pattern

Count the number of matches in a string

Extract the content of matches

Replace matches with new values

Package: **stringr**

Functions: `stringr::str_detect()`, `str_subset()`, `str_count()`, `str_extract()`, `str_replace()`

Operator precedence and parenthesis

What does ab+ match? Does it match “a” followed by one or more “b”s, or does it match “ab” repeated any number of times?

And ^a|b$? Does it match the complete string “a” or the complete string “b”, or does it match a string starting with “a” or a string ending with “b”?

The answer to these questions is provided by **operator precedence**: quantifiers have high precedence, while alternation has low precedence. You can use parentheses to override the usual order.

So what is the meaning of ab+?

And ^a|b$?

Operator precedence and parenthesis

What does ab+ match? Does it match “a” followed by one or more “b”s, or does it match “ab” repeated any number of times?

And ^a|b$? Does it match the complete string “a” or the complete string “b”, or does it match a string starting with “a” or a string ending with “b”?

The answer to these questions is provided by **operator precedence**: quantifiers have high precedence, while alternation has low precedence. You can use parentheses to override the usual order.

So what is the meaning of ab+? a(b)+

And ^a|b$? (^a)|(b$)

Suggestion: since it’s unlikely you remember the precedence rules for regexes, I suggest to use parentheses.

Stringr functions

We will work with six main verbs that work with patterns:

`str_detect()`	detect a pattern – returns T/F
`str_count()`	count the number of matches
`str_subset()`	returns only the strings that contain the pattern
`str_extract()`	extract the text of the first match, returns a vector
`str_replace()`	replaces the first matched pattern

Detect matches

`str_detect()` Detect the presence or absence of a pattern in a string

```
str_detect(string, pattern, negate = FALSE)
```

`string`	A character vector.
`pattern`	Pattern to look for (regular expression).
`negate`	If TRUE, return non-matching elements.

This function returns a logical vector with the same length as the input.

Detect matches

```
> x <- c("apple", "banana", "pear")
> str_detect(x, "e")
[1] TRUE FALSE TRUE
```

Argument `negate`:

```
> str_detect(x, "e", negate = TRUE)
[1] FALSE TRUE FALSE
```

Using the pipe operator

```
> x|>str_detect("e")
[1] TRUE FALSE TRUE
```

Detect matches

Since `str_detect()` returns a logical vector of the same length as the initial vector, it pairs well with `filter()`.

For example, consider the babynames data contained in the **babynames** package. Let us select all the names containing "x":

```
> install.packages("babynames")
> library(babynames)
> View(babynames)
> babynames |>
+ filter(str_detect(name, "x")) |>
+ count(name, sort = TRUE)
# A tibble: 974 × 2
name n
<chr> <int>
1 Maxie 245
2 Alex 237
3 Alexander 226
```

Detect matches

`str_detect()` is also frequently used with `summarise()` by pairing it with `sum()` and `mean()`:

`sum(str_detect(x, pattern))` reports the number of observations that match

`sum(str_detect(x, pattern))` reports the proportion of observations that match.

For example, let us compute the total number and proportion of baby names that contain «x» by year:

```
> babynames |>
+ group_by(year) |>
+ summarise(sumx = sum(str_detect(name, "x")), propx = mean(str_detect(name, "x")))
# A tibble: 138 × 3
year sumx propx
<dbl> <int> <dbl>
1 1880 13 0.0065
2 1881 17 0.00879
```

Counting

`str_count()` Count the number of matches in a string.

```
str_count(string, pattern = "")
```

`string`	A character vector.
`pattern`	Pattern to look for (regular expression).

Counting

Assume you want to count how many *a*'s are present in each of the three words in *x*

```
> x
[1] "apple" "banana" "pear"
> str_count(x, "a")
[1] 1 3 1
```

Using the pipe operator

```
> x |> str_count("a")
[1] 1 3 1
```

Notice the output using the default value for the argument `pattern`

```
> str_count(x) [1] 5 6 4
```

Counting

It is natural to use `str_count()` with `mutate()`.

In the babynames dataframe, let us create variables containing the number of vowels and consonants for each name:

```
> babynames |>
+ count(name) |>
+ mutate(
+      vowels = str_count(name, "[aeiou]"),
+      consonants = str_count(name, "[^aeiou]")
+ )
# A tibble: 97,310 × 4
name n vowels consonants
<chr> <int> <int> <int>
1 Aaban 10    2      3
2 Aabha 5     2      3
```

Is there anything wrong in this computation???

Exercise

Write a proper code to answer the following questions:

1) How many common words start with t?
2) What proportion of common words end with a vowel?
3) Count how many vowels are contained in each word in `stringr::words`.
4) How many vowels are contained per word on average?

Subsetting

`str_subset()` Keep strings matching a pattern.

```
str_subset(string, pattern, negate = FALSE)
```

`string`	A character vector.
`pattern`	Pattern to look for (regular expression).
`negate`	If TRUE, return non-matching elements.

Subsetting

Assume you want to select all the words that end with *x*

```
> str_subset(words, "x$")
[1] "box" "sex" "six" "tax"
```

Notice that this is equivalent to using `str_detect` and logical subsetting:

```
> words[str_detect(words, "x$")]
[1] "box" "sex" "six" "tax"
```

We could also use the pipe operator

```
> words |> str_subset("x$")
[1] "box" "sex" "six" "tax"
```

Extracting

`str_extract()` Extract matching patterns from a string. `str_extract()` extracts the text corresponding to the first match, returning a character vector. `str_extract_all()` extracts all matches and returns a list of character vectors.

```
str_extract(string, pattern)

str_extract_all(string, pattern, simplify = FALSE)
```

`string`	A character vector.
`pattern`	Pattern to look for (regular expression).
`simplify`	If `FALSE`, the default, returns a list of character vectors. If `TRUE` returns a character matrix.

Extracting

Assume I want to extract letters *n* or *l* from the words in vector *x*.

```
> str_extract(x, "[nl]")
[1] "l" "n" NA
```

Using the pipe operator

```
> x %>% str_extract("[nl]")
[1] "l" "n" NA
```

Compare with the use of `str_subset()`

```
> str_subset(x, "[nl]")
[1] "apple"  "banana"
```

Extracting

Notice that `str_extract()` extracts the first match only. This is common in **stringr** functions, as working with a single match allows to use much simpler data structures. To get all matches, use `str_extract_all()`. It returns a list

```
> str_extract_all(x, "[nl]")
[[1]]
[1] "l"

[[2]]
[1] "n" "n"

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

Using `simplify = TRUE`, `str_extract_all()` will return a matrix with short matches expanded to the same length as the longest

```
> str_extract_all(x, "[nl]", simplify = TRUE)
     [,1] [,2]
[1,] "l"  ""
[2,] "n"  "n"
[3,] ""   ""
```

Replacing

`str_replace()` Replace matched patterns in a string.

```
str_replace(string, pattern, replacement)

str_replace_all(string, pattern, replacement)
```

`string`	A character vector.
`pattern`	Pattern to look for (regular expression).
`replacement`	A character vector of replacements. Should be either length one, or the same length as string or pattern.

Replacing

Assume you want to replace the first vowel with a dash

```
> x
[1] "apple"  "banana" "pear"
> str_replace(x, "[aeiou]", "-")
[1] "-pple"  "b-nana" "p-ar"
```

Replace all vowels with dash

```
> str_replace_all(x, "[aeiou]", "-")
[1] "-ppl-"  "b-n-n-" "p--r"
```

Using the pipe operator

```
> x %>% str_replace_all("[aeiou]", "-")
[1] "-ppl-"  "b-n-n-" "p--r"
```

Replacing

These functions are vectorised over pattern and replacement

```
> xx = c("1 horse 1", "2 cars", "3 people")
> str_replace(xx, c("1", "2", "3"), c("one", "two", "three"))
[1] "one horse 1"  "two cars"     "three people"

> str_replace_all(xx, c("1", "2", "3"),c("one", "two", "three"))
[1] "one horse one" "two cars"      "three people"
```

Other functions

`str_locate()` and `str_locate_all()` give the starting and end in positions of each match.

`str_which()` returns an integer vector giving the positions of the strings that match.

`str_split()` splits the string based on a pattern and returns a list or a matrix

Exercise

Consider the sample character vectors `stringr::sentences`. This is a collection of "Harvard sentences" used for standardised testing of voice.

```
> length(sentences)
[1] 720
> head(sentences)
[1] "The birch canoe slid on the smooth planks."
[2] "Glue the sheet to the dark blue background."
[3] "It's easy to tell the depth of a well."
[4] "These days a chicken leg is a rare dish."
[5] "Rice is often served in round bowls."
[6] "The juice of lemons makes fine punch."
```

We want to find all the sentences that contain a color and extract the corresponding colors.

1) Create a vector (named colors) of color names (containing the colors red, orange, yellow, green, blue, and purple)
2) Turn the vector colors into a singular regular expression
3) Select the sentences that contain a color
4) Extract the color from the sentences identified at point 3.

Exercises for you

Exercise 1. With reference to the character vector `stringr::words`, write proper code to perform the following tasks:

1) Find all words that start with g.
2) Find all the words that end with g.
3) Find all the words that start or end with g. Try solving the challenge by using both a single regular expression and a combination of multiple `str_detect()` calls.
4) Find all words that start with a vowel and end with a consonant
5) What word has the highest number of vowels? What word has the highest proportion of vowels?

Exercises for you

Exercise 2. Reconsider the exercise on Slide 23.

1) Can you identify any issue in the identified sentences (containing a color)? Explain.
2) Correct the identified issue and select the sentences that contain a color. Only match colors in which the entire word is the name of the color and not allow matches with words that have the name of a color inside. Hint: the use of boundaries `\b` might be useful in defining the regular expression.
3) Extract the color from the sentences identified at point 2.
4) How many sentences contain more than one match?
5) Print the sentences containing more than one match.
6) Extract all the matches from the sentences identified at point 5.