The ABC of Computational Text Analysis

#6 Learning Regular Expressions

Faculty of Humanities and Social Sciences
University of Lucerne

01 April 2022

Recap last Lecture

- well solved assignment #1 🎊

  example solution

- counting words 🔢

  particular or all words

- preprocessing and cleaning 🧼

Outline

- introduction regular expression ✨
- practicing RegEx 🎢

Text as Pattern

Formal Search Patterns

How to extract **any** email address in a text collection?

```
Please contact us via info@organization.org.
---
For specific questions ask Mrs. Green (a.green@mail.com).
---
Reach out to support@me.ch
```

Solution: Write a single pattern to match any valid email all

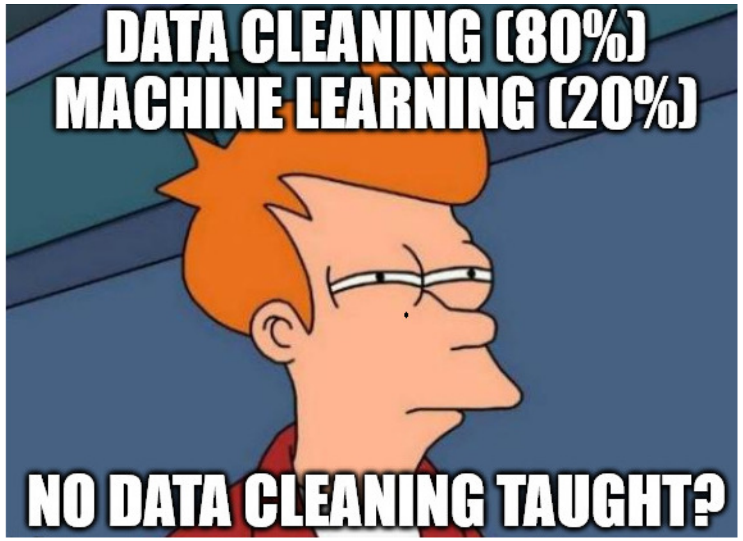
```
[\w.-_]+@[\w-_]+\.[a-z]{2,}     # matches any valid email address
```

What are patterns for?

- finding 🔎
- extracting 🛠️
- removing/cleaning 🗑️
- replacing 🔁

… any parts in texts

Data Cleaning is paramount!



What is RegEx?

RegEx = Regular Expressions = Patterns

- **literal** string of characters

  word, phrases, dates etc.

- highly flexible **meta** expressions

  e.g., `\w` represents alphanumeric characters

- `[Cc]o+l` $\longrightarrow$ Col, col, Cool, coool …

- akin to regular languages

Finding + Extracting

globally search for **r**egular **e**xpression and **p**rint (grep)

- tool to filter/keep certain lines only
- allow extended regex patterns

  use `egrep` instead of `grep`

```
egrep 'yes' file.txt        # search in a specific file
egrep -r 'yes' folder       # search recursively within folder

egrep 'yes' *.txt           # keep lines containing pattern (yes)
egrep -i 'yes' *.txt        # dito, ignore casing (Yes, yes, YES ...)
egrep -v 'noisy' *.txt      # do NOT keep lines containing noisy

# extract raw match only to allow for subsequent counting
egrep -o 'only' *.txt       # print match only, not entire line
egrep -h 'only' *.txt       # suppress file name
```

Quantifiers

repeat preceding character `x` times

- `?` zero or one
- `+` one or more
- `*` zero or any number
- `{n}`, `{m,n}` a specified number of times

```
egrep -r "Bundesrath?es"          # match old and new spelling of
egrep -r "aa+"                    # match two or more "a"
egrep -r "e{2}"                   # match sequence of two "e"
```

⚠️ Do not confuse regex with Bash wildcards!

Character Sets

- `[...]` any of these character

  any vowel: `[auoei]`

  any digit: `[0-9]`

  any letter: `[A-Za-z]`

- `[^...]` any character but none of these

  anything but the vowels: `[^auoei]`

```
# match the capitalized and non-capitalized form
egrep -r '[Gg]rüne'

# match sequences of 3 vowels
egrep -r [aeiou]{3}

# extract all bigrams (sequence of two words)
egrep -rohi '[a-z]+ [a-z]+'
```

Special Symbols

- `.` any character (excl. newline)
- `\` escaping to match literal

  `\.` means the literal `.` instead of “any symbol”
- `\w` any alpha-numeric character

  same as `[A-Za-z0-9_]`
- `\s` any whitespace (space, newline, tab)

  same as `[ \t\n]`

```
# match anything between brackets
egrep -r '\(.*\)'
```

The power of `.*` ...

matches ***any character any times***

More complex Examples

```
# extract basename of URLs
egrep -ro "www\.[\w-_]+\.[a-z]{2,}"

# extract valid email adresses
egrep -ro "[\w.-_]+@[\w-_]+\.[a-z]{2,}" */*.txt
```

Combining RegEx with Frequency Analysis

something actually useful

```
# count political areas by looking up words ending with "politik"
egrep -rioh '\w*politik' */*.txt | sort | uniq -c | sort -h

# count ideologies/concepts by looking up words ending with "ismus"
egrep -rioh '\w*ismus' */*.txt | sort | uniq -c | sort -h
```

Start simple, add complexity subsequently.

Replacing + Removing

stream editor (sed)

- advanced find + replace using regex

  `sed "s/WHAT/WITH/g" file.txt`

- `sed` replaces any sequence, `tr` only single symbols

```
echo "hello" | sed "s/llo/y/g"      # replace "llo" with a "y"

# by setting the g flag in "s/llo/y/g",
# sed replaces all occurences, not only the first one
```

Contextual Replacing

reuse match with grouping

- define a group with parentheses `(group_pattern)`
- `\1` equals the expression inside first pair of parentheses
- `\2` expression of second pair
- ...

```
# swap order of name (last first -> first last)
echo "Lastname Firstname" | sed -E "s/(.*) (.*)/\2 \1/"

# matching also supports grouping
# match any pair of digits (two identical digits)
egrep -r "([0-9])\1([0-9])\2"
```

More Meta-Symbols

- `\b` word boundary

  `word\b` does not match `words`

- `^` begin of line and `$` end of line

  `^A` matches only `A` at line start

- `|` disjunction (OR)

  `(Mr|Mrs|Mr\.|Mrs\.) Green` matches alternatives

Greediness Trap

- greedy ~ match the longest string possible
- quantifiers `*` or `+` are greedy
- non-greedy by excluding some symbols

  `[^EXCLUDE_SYMBOLS]` *instead of* `.*`

```
# greedy: an apple, other apple
echo 'an apple, other apple' | egrep 'a.*apple'

# non-greedy: an apple
echo 'an apple, other apple' | egrep 'a[^,]*apple'
```

Assignment #2 ✍️

- get/submit via OLAT

    starting tomorrow

    deadline 9 April 2022, 23:59

- use forum on OLAT

    subscribe to get notifications

- ask friends for support, not solutions

In-class: Game

1. Make sure that your local copy of the Github repository KED2022 is up-to-date with `git pull`. Go to the party programmes in `materials/party_programmes/txt`.
2. Use `egrep` to extract all uppercased words like `UNO, OECD, SP` and count their frequency.
3. Send me a private chat message with the most frequent abbreviation from 2).

```
# Some not so random hints
piping with |
sort
uniq -c
egrep -roh */*.txt
```

In-class: Exercises I

1. Update your local copy of the GitHub repository KED2022 with `git pull`. Go to the party programmes in `KED2022/materials/data/swiss_party_programmes/txt`.
2. Use `egrep` to extract all uppercased words that are abbreviations in most cases (e.g., UNO, SVP, SP).
3. Use `egrep` to extract words following any of these strings: `der die das`.
4. Do the self-check on the next slide.
5. Use `sed -E` to remove the table of content, the footer and the page number in the programme of the Green Party. Check the corresponding PDF to get a visual impression and test your regular expression with `egrep` first to see if you match the correct parts in the document.

In-class: Self-Check

equivalent patterns

```
a+ == aa*                  # "a" once or more than once
a? == (a|_)                # "a" once or nothing
a{3} == aaa                # three "a"
a{2,3} == (aa|aaa)         # two or three "a"
[ab] == (a|b)              # "a" or "b"
[0-9] == (0|1|2|3|4|5|6|7|8|9)  #any digit
```

In-class: Exercise II

1. Since you know about RegEx, we can use a more sophisticated tokenizer to split a text into words. What is the difference between the old and new approach? Test it and check the helper page with `man`.

```
# new, improved approach
cat text.txt | tr -sc "[a-zäöüA-ZÄÖÜ0-9-]" "\n"

# old approach
cat text.txt | tr ' ' '\n'
```

In-class: Exercise III

1. Count all the bigrams (sequence of two words) using character sets and quantifiers. What about trigrams (three words)?
2. Extract the words following numbers (also consider numbers like: `1'000, 1,000 or 5%`). Then, count all the words while excluding the numbers themselves. Hint: Pipe another grep to remove the digits.
3. You are ready to come up with your own patterns…

More Resources

required

- Ben Schmidt. 2019. Regular Expressions. online

highly recommended

- Nikolaj Lindberg. egrep for Linguists. online

Have a nice
Easter break!