

# The ABC of Computational Text Analysis

## #6 LEARNING REGULAR EXPRESSIONS

Alex Flückiger  
Faculty of Humanities and Social Sciences  
University of Lucerne

30 March 2024

# Recap last Lecture

- well-solved assignment #1 
- example solution
- counting words 
- particular words or entire vocabulary
- preprocessing and cleaning 

# Outline

- introducing regular expression ✨
- practicing the writing of patterns 🎢

# Text as Pattern

# Searching with Patterns

How to extract all email addresses 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 as soon as possible.

...

👉 **Solution:** Write a single pattern to match any valid email address

```
[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,} # case-insensitive
```

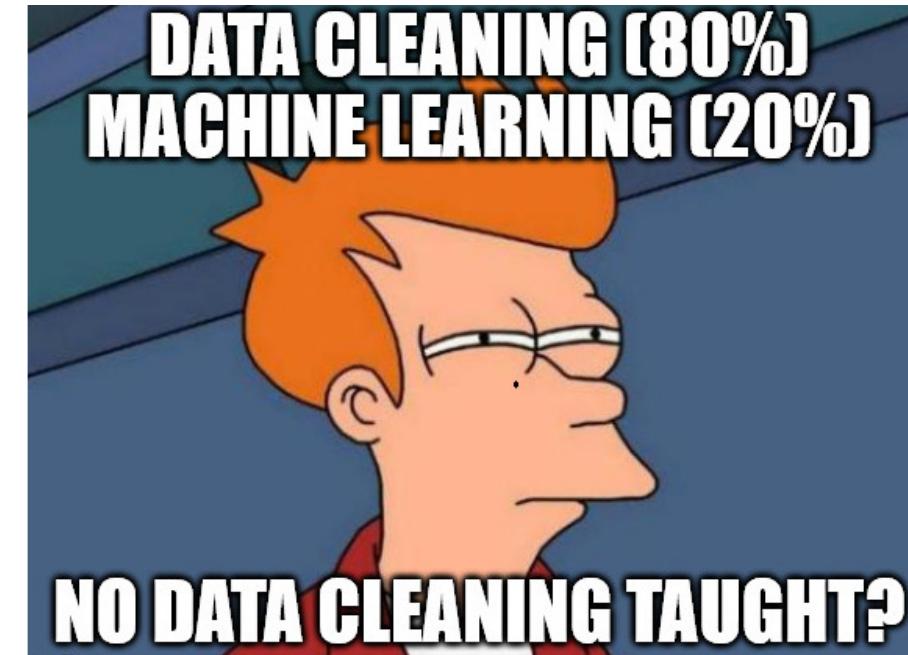
# Data Cleaning is paramount!

Removing parts from real-word data

## INHALTSVERZEICHNIS

GENDERNEUTRALE SPRACHE	4
EINLEITUNG	5
WOFÜR WIR UNS EINSETZEN	6
1. DIE GRÜNEN SIND DIE PARTEI DER UMWELT	1
1.1 FÜR DIE UMSETZUNG DES KLIMAABKOMMENS VON PARIS	1
1.2 FÜR DIE BESCHLEUNIGUNG DER GRÜNEN ENERGIEWENDE	2
1.3 FÜR EINE UMWELTGERECHTE UND MENSCHENFREUNDLICHE MOBILITÄT	3
1.4 FÜR DEN SCHUTZ DER NATUR UND DEN ERHALT DER BIODIVERSITÄT	5
1.5 FÜR EINE HAUSHÄLTERISCHE BODENNUTZUNG UND EINE KONSEQUENTE RAUMPLANUNG	6

*The table of content may be removed*



# When using Patterns?

- finding 
- extracting 
- removing/cleaning 
- replacing 

**... specific parts in texts**

# Use `egrep` with Regular Expressions

**extended globally search for regular expression and print (egrep)**

```
# check a regular expression quickly
echo "check my first pattern" | egrep "[0-9]"
echo "check my 1st pattern" | egrep "[0-9]"
```

# egrep in Practice

## egrep filters/keeps matching lines only

```
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) a  
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 instead of entire li  
egrep -h "only" *.txt        # suppress file name
```

# What are Regular Expressions (RegEx)?

RegEx builds on two classes of symbols

- **literal** characters and strings
  - letters, digits, words, phrases, dates etc.
- **meta** expressions with special meaning
  - e.g., `\w` represents alphanumeric characters
  - `[Cc)o+l` → Col, col, Cool, coool ...
- akin to regular languages

# 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 word with optional "h"  
egrep -r "a+"                      # match one or more "a"  
egrep -r "e{2}"                    # match sequence of two "e"
```



Do not confuse regex with Bash wildcards!

# Character Sets

- [ . . . ] any of the characters between brackets
  - any vowel: [auoei]
  - any digit: [0-9]
  - any letter: [A-Za-z]
- [^ . . . ] any character except those (**negation**)
  - anything but the vowels: [^auoei]

```
egrep -r "[Gg]rüne" # match both `grüne` and `Grüne`  
egrep -r "[aeiou]{3}" # match sequences of 3 vowels  
egrep -rohi "[a-z]+ [a-z]+" # extract bigrams (sequence of two word
```

# Special Symbols

- `.` matches any character (excl. newline)
- `\` escapes to match literal
  - `\.` means the literal `.` instead of “any symbol”
- `\w` matches any alpha-numeric character
  - same as `[A-Za-z0-9_]`
- `\s` matches any whitespace (space, newline, tab)
  - same as `[ \t\n]`

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

The power of . \* 💪

Match *any character any times*

# More Complex Examples

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

# extract valid email addresses (case-insensitive)
egrep -iro "[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[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.

# In-class: Exercise

1. Use the command line to navigate to the local copy of the Github repository KED2024 and make sure it is up-to-date with `git pull`. Change in to the directory `KED2024/materials/data/swiss_party_programmes.txt`.
2. Use `egrep` to extract all uppercased words like `UNO`, `OECD`, `SP` and count their frequency (check hint below ).
3. Similarly, use `egrep` to extract all plural nouns with female endings (e.g., `Schweizerinnen`). The pattern matches words starting with an uppercase letter, ending with `innen`, and any letter in between. Do the same for the male forms. Is there a qualitative or a quantitative difference between the gendered forms?

```
# write a pattern
egrep -roh "YOUR_PATTERN" **/*.txt | sort | uniq -c | sort -h
```

# In-class: Solution

2. Use egrep to extract all uppercased words like UNO, OECD, SP and count their frequency.

```
egrep -roh "[A-Z]{2,}" **/*.txt | sort | uniq -c | sort -h
```

3. Use egrep to extract all plural nouns with female endings (e.g. Schweizerinnen). The pattern matches words that start with an uppercase letter, ending with `innen`, and any letter in between. Do the same for the male forms. Is there a qualitative or a quantitative difference between the gendered forms?

```
egrep -roh "[A-Z][a-z]+innen\b" **/*.txt | sort | uniq -c | sort -h
```

```
egrep -roh "[A-Z][a-z]+er\b" **/*.txt | sort | uniq -c | sort -h
```

(there is no way with regular expression to extract only nouns of the male form but not `Wasser` and the like. For this, you have to use some kind of machine learning.)

# 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 occurrences, 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 two identical digits
egrep -r "([0-9])\1"
```

# More Meta-Symbols

- `\b` matches word boundary  
`word\b` does not match `words`
- `^` matches begin of line and `$` end of line  
`^A` matches only `A` at line start
- `|` is a 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 6 April 2024, 23:59
- use forum on OLAT
  - subscribe to get notifications
- ask friends for support, not solutions

# In-class: Exercises I

1. Use egrep to extract capitalized words and count them. What are the most frequent nouns?
2. Use egrep to extract words following any of these strings: der die das.  
Hint: Use a disjunction.
3. Do the self-check on the next slide.
4. 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: Solution I

1. Use egrep to extract capitalized words and count them. What are the most frequent nouns?

```
egrep -roh "[A-Z][a-z]+" */*.txt | sort | uniq -c | sort -h
```

2. Use egrep to extract words following any of these strings: der die das.

Hint: Use a disjunction.

```
egrep -roh "(der|die|das) \w+" */*.txt
```

3. 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.

```
cat gruene_programme_2019.txt | sed "1,192d" | sed -E  
"s/^Wahlplattform.*2023$/g" | sed -E "s/^[\0-9]+$/g"
```

# 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. 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...

# In-class: Solution II

1. Count all the bigrams (sequence of two words) using character sets and quantifiers. What about trigrams (three words)?

```
egrep -hoir "\b[a-z]+ [a-z]+\b" | sort | uniq -c | sort -h
```

```
egrep -hoir "\b[a-z]+ [a-z]+\b" | sort | uniq -c | sort -h
```

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.

```
egrep -rhoi "[0-9][0-9,%]+ [a-z]+" | egrep -io "[a-z]+" | sort | uniq -c | sort -h
```

Alternative: `egrep -rhoi "[0-9][0-9,%]+ [a-z]+" | sed -E "s/[0-9][0-9,%]+//g" | sort | uniq -c | sort -h`

# In-class: Exercise III

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-zAÖÜöü-9-]" "\n"

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

# More Resources

## Helpful resources for assignment

- Ben Schmidt. 2019. [Regular Expressions](#).
- [Cheatsheet](#) of this course
- online regular expression editor  
[regex101](#) to write and check patterns

## recommended reading

- Nikolaj Lindberg. [egrep for Linguists](#).



# Questions?