# Big Data & Automated Content Analysis
# Week 5 – Wednesday: »Working with text«

Damian Trilling

d.c.trilling@uva.nl
@damian0604
www.damiantrilling.net

3 March 2021

Afdeling Communicatiewetenschap
Universiteit van Amsterdam

# Today

Bottom-up vs. top-down

Approaches to working with text

- The toolbox
- From test to large-scale analysis

ACA using regular expressions

- What is a regexp?
- Using a regexp in Python

# Everything clear from last week?

# Bottom-up vs. top-down

Automated content analysis can be either bottom-up (inductive, explorative, pattern recognition, ...) or top-down (deductive, based on a-priori developed rules, ...). Or in between.

# The ACA toolbox

| | **Methodological approach** | | |
|---|---|---|---|
| | *Counting and Dictionary* | *Supervised Machine Learning* | *Unsupervised Machine Learning* |
| **Typical research interests and content features** | visibility analysis<br>sentiment analysis<br>subjectivity analysis | frames<br>topics<br>gender bias | frames<br>topics |
| **Common statistical procedures** | string comparisons<br>counting | support vector machines<br>naive Bayes | principal component analysis<br>cluster analysis<br>latent dirichlet allocation<br>semantic network analysis |


Boumans and Trilling, 2016

# Bottom-up vs. top-down

## Bottom-up

- Count most frequently occurring words
- Maybe better: Count combinations of words ⇒ Which words co-occur together?

We *don't* specify what to look for in advance

## Top-down

- Count frequencies of pre-defined words
- Maybe better: patterns instead of words

We *do* specify what to look for in advance

# Bottom-up vs. top-down

## Bottom-up

- Count most frequently occurring words
- Maybe better: Count combinations of words ⇒ Which words co-occur together?

We *don't* specify what to look for in advance

## Top-down

- Count frequencies of pre-defined words
- Maybe better: patterns instead of words

We *do* specify what to look for in advance

# A simple bottom-up approach

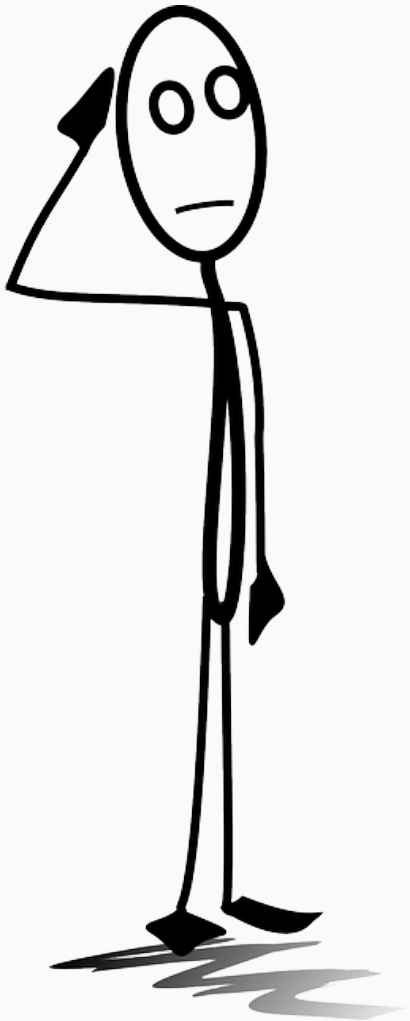
```
1  from collections import Counter
2
3  texts = ["I really really really love him, I do", "I hate him"]
4
5  for t in texts:
6      print(Counter(t.split()).most_common(3))
```

```
1  [('really', 3), ('I', 2), ('love', 1)]
2  [('I', 1), ('hate', 1), ('him', 1)]
```

# A simple top-down approach

```
1  texts = ["I really really really love him, I do", "I hate him"]
2  features = ['really', 'love', 'hate']
3
4  for t in texts:
5      print(f"\nAnalyzing '{t}':")
6      for f in features:
7          print(f"{f} occurs {t.count(f)} times")
```

```
1  Analyzing 'I really really really love him, I do':
2  really occurs 3 times
3  love occurs 1 times
4  hate occurs 0 times
5
6  Analyzing 'I hate him':
7  really occurs 0 times
8  love occurs 0 times
9  hate occurs 1 times
```



*When would you use which approach?*

# Approaches to working with text

# Approaches to working with text

## The toolbox

# The toolbox

## Slicing

`mystring[2:5]` to get the characters with indices 2,3,4

## String methods

- `.lower()` returns lowercased string
- `.strip()` returns string without whitespace at beginning and end
- `.find("bla")` returns index of position of substring “bla” or -1 if not found
- `.replace("a","b")` returns string where "a" is replaced by "b"
- `.count("bla")` counts how often substring “bla” occurs

Use tab completion for more!

# The toolbox

**Regular expressions**

(today)

# Approaches to working with text

## From test to large-scale analysis

# General approach

1. Take a single string and test your idea

```
1 t = "This is a test test test."
2 print(t.count("test"))
```

2a. You'd assume it to return 3. If so, scale it up:

```
1 results = []
2 for t in listwithallmytexts:
3     r = t.count("test")
4     print(f"{t} contains the substring {r} times")
5     results.append(r)
```

2b. If you *only* need to get the list of results, a list comprehension is more elegant:

```
1 results = [t.count("test") for t in listwithallmytexts]
```

# General approach

Test on a single string, then make a for loop or list comprehension!

## Own functions

If it gets more complex, you can write your own function and then use it in the list comprehension:

```
1 def mycleanup(t):
2     # do sth with string t here, create new string t2
3     return t2
4
5 results = [mycleanup(t) for t in allmytexts]
```

# General approach

Test on a single string, then make a for loop or list comprehension!

## Own functions

If it gets more complex, you can write your own function and then use it in the list comprehension:

```
1  def mycleanup(t):
2      # do sth with string t here, create new string t2
3      return t2
4
5  results = [mycleanup(t) for t in allmytexts]
```

# Pandas string methods as alternative

If you select column with strings from a pandas dataframe, pandas offers a collection of string methods (via `.str.`) that largely mirror standard Python string methods:

```
1 df['newcoloumnwithresults'] = df['columnwithtext'].str.count("bla")
```

## To pandas or not to pandas for text?

Partly a matter of taste.

Not-too-large dataset with a lot of extra columns? Advanced statistical analysis planned? Sounds like pandas.

It's mainly a lot of text? Wanna do some machine learning later on anyway? It's large and (potentially) messy? Doesn't sound like pandas is a good idea.

# Pandas string methods as alternative

If you select column with strings from a pandas dataframe, pandas offers a collection of string methods (via `.str.`) that largely mirror standard Python string methods:

```
1 df['newcoloumnwithresults'] = df['columnwithtext'].str.count("bla")
```

**To pandas or not to pandas for text?**

Partly a matter of taste.

Not-too-large dataset with a lot of extra columns? Advanced statistical analysis planned? Sounds like pandas.

It's mainly a lot of text? Wanna do some machine learning later on anyway? It's large and (potentially) messy? Doesn't sound like pandas is a good idea.

# Regular expressions

Automated content analysis using regular expressions

# Regular expressions

## What is a regexp?

# Regular Expressions: What and why?

## What is a regexp?

- a *very* widespread way to describe patterns in strings
- Think of wildcards like * or operators like `OR`, `AND` or `NOT` in search strings: a regexp does the same, but is *much* more powerful
- You can use them in many editors (!), in the Terminal, in STATA ... and in Python

# Regular Expressions: What and why?

## What is a regexp?

- a *very* widespread way to describe patterns in strings
- Think of wildcards like * or operators like `OR`, `AND` or `NOT` in search strings: a regexp does the same, but is *much* more powerful
- You can use them in many editors (!), in the Terminal, in STATA ... and in Python

# Regular Expressions: What and why?

## What is a regexp?

- a *very* widespread way to describe patterns in strings
- Think of wildcards like * or operators like `OR`, `AND` or `NOT` in search strings: a regexp does the same, but is *much* more powerful
- You can use them in many editors (!), in the Terminal, in STATA . . . and in Python

# An example

## From last week's task

- We wanted to remove everything but words from a tweet
- We did so by calling the `.replace()` method
- We could do this with a regular expression as well:
  `[^a-zA-Z]` would match anything that is not a letter

# Basic regexp elements

## Alternatives

`[TtFf]` matches either T or t or F or f

`Twitter|Facebook` matches either Twitter or Facebook

`.` matches any character

## Repetition

`*` the expression before occurs 0 or more times

`+` the expression before occurs 1 or more times

# Basic regexp elements

## Alternatives

`[TtFf]` matches either T or t or F or f

`Twitter|Facebook` matches either Twitter or Facebook

`.` matches any character

## Repetition

`*` the expression before occurs 0 or more times

`+` the expression before occurs 1 or more times

# regexp quizz

## Which words would be matched?

1. `[Pp]ython`
2. `[A-Z]+`
3. `RT ?:? @[a-zA-Z0-9]*`

# regexp quizz

## Which words would be matched?

1. `[Pp]ython`
2. `[A-Z]+`
3. `RT ?:? @[a-zA-Z0-9]*`

# regexp quizz

## Which words would be matched?

1. `[Pp]ython`
2. `[A-Z]+`
3. `RT ?:?  @[a-zA-Z0-9]*`

# What else is possible?

See the table in the book!

# Regular expressions

## Using a regexp in Python

# How to use regular expressions in Python

## The module `re`*

`re.findall("[Tt]witter|[Ff]acebook",testo)` returns a list with all occurances of Twitter or Facebook in the string called `testo`

`re.findall("[0-9]+[a-zA-Z]+",testo)` returns a list with all words that start with one or more numbers followed by one or more letters in the string called `testo`

`re.sub("[Tt]witter|[Ff]acebook","a social medium",testo)` returns a string in which all all occurances of Twitter or Facebook are replaced by "a social medium"

Use the less-known but more powerful module `regex` instead to support all dialects used in the book

# How to use regular expressions in Python

## The module `re`*

`re.findall("[Tt]witter|[Ff]acebook",testo)` returns a list with all occurances of Twitter or Facebook in the string called `testo`

`re.findall("[0-9]+[a-zA-Z]+",testo)` returns a list with all words that start with one or more numbers followed by one or more letters in the string called `testo`

`re.sub("[Tt]witter|[Ff]acebook","a social medium",testo)` returns a string in which all all occurances of Twitter or Facebook are replaced by "a social medium"

Use the less-known but more powerful module `regex` instead to support all dialects used in the book

# How to use regular expressions in Python

## The module re

`re.match(" +([0-9]+) of ([0-9]+) points",line)` returns `None` unless it *exactly* matches the string `line`. If it does, you can access the part between () with the `.group()` method.

Example:

```
1  line="           2 of 25 points"
2  result=re.match(" +([0-9]+) of ([0-9]+) points",line)
3  if result:
4     print ("Your points:",result.group(1))
5     print ("Maximum points:",result.group(2))
```

Your points: 2

Maximum points: 25

# Possible applications

## Data preprocessing

- Remove unwanted characters, words, ...
- Identify *meaningful* bits of text: usernames, headlines, where an article starts, ...
- filter (distinguish relevant from irrelevant cases)

# Possible applications

## Data analysis: Automated coding

- Actors
- Brands
- links or other markers that follow a regular pattern
- Numbers (!)

# Example 1: Counting actors

```
1  import re, csv
2  from glob import glob
3  count1_list=[]
4  count2_list=[]
5  filename_list = glob("/home/damian/articles/*.txt")
6
7  for fn in filename_list:
8     with open(fn) as fi:
9        artikel = fi.read()
10       artikel = artikel.replace('\n',' ')
11
12       count1 = len(re.findall('Israel.*(minister|politician.*|[Aa]
             uthorit)',artikel))
13       count2 = len(re.findall('[Pp]alest',artikel))
14
15       count1_list.append(count1)
16       count2_list.append(count2)
17
18 output=zip(filename_list,count1_list, count2_list)
19 with open("results.csv", mode='w',encoding="utf-8") as fo:
20     writer = csv.writer(fo)
21     writer.writerows(output)
```

# Example 2: Which number has this Lexis Nexis article?

```
                                All Rights Reserved

                                2 of 200 DOCUMENTS

                                   De Telegraaf

                                21 maart 2014 vrijdag

Brussel bereikt akkoord aanpak probleembanken;
ECB krijgt meer in melk te brokkelen

SECTION: Finance; Blz. 24
LENGTH: 660 woorden

BRUSSEL  Europa heeft gisteren op de valreep een akkoord bereikt
over een saneringsfonds voor banken. Daarmee staat de laatste
```

# Example 2: Check the number of a lexis nexis article

```
                                All Rights Reserved

                                 2 of 200 DOCUMENTS

                                    De Telegraaf

                                 21 maart 2014 vrijdag

Brussel bereikt akkoord aanpak probleembanken;
ECB krijgt meer in melk te brokkelen

SECTION: Finance; Blz. 24
LENGTH: 660 woorden

BRUSSEL  Europa heeft gisteren op de valreep een akkoord bereikt
over een saneringsfonds voor banken. Daarmee staat de laatste
```

```
for line in tekst:
    matchObj=re.match(r" +([0-9]+) of ([0-9]+) DOCUMENTS",line)
    if matchObj:
```

# Practice yourself!

Let's take some time to write some regular expressions. Write a script that

- extracts URLS form a list of strings
- removes everything that is not a letter or number from a list of strings

(first develop it for a single string, then scale up)

More tips: http://www.pyregex.com/

# Next meetings

## Friday

Write your own ACA script!

Let's take a large dataset on Nørregaard et al., 2019. It's really large and may take some time to download and unpack (1 GB compressed, 10 GB unpacked)! It can be wise to already download and unpack it, see https://github.com/damian0604/bdaca/blob/master/12ec/week05/exercises/exercise.md.

## TAKE HOME EXAM

Handed out after Friday's meeting

Deadline: Tuesday, 23.59

# References


Boumans, J. W., & Trilling, D. (2016). Taking stock of the toolkit: An overview of relevant autmated content analysis approaches and techniques for digital journalism scholars. *Digital Journalism*, *4*(1), 8–23. https://doi.org/10.1080/21670811.2015.1096598

Nørregaard, J., Horne, B. D., & Adali, S. (2019). *NELA-GT-2018* [Dataset for article "NELA-GT-2018: A Large Multi-Labelled News Dataset for the Study of Misinformation in News Articles". (2019-01-15)]. Harvard Dataverse. https://doi.org/10.7910/DVN/ULHLCB