

# Big Data and Automated Content Analysis

## Part I+II

Week 14 – Wednesday  
»Looking back and froward«

Damian Trilling

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

Afdeling Communicatiewetenschap  
Universiteit van Amsterdam

13 May 2020

# Today

## ① Looking back

Putting the pieces together  
A good workflow

## ② Looking forward

Techniques we did not cover  
Neural Networks

## ③ The INCA project

Scaling up Content Analysis  
The INCA architecture

## ④ Final steps

Looking back

**Putting the pieces together**

# First: Our epistemological underpinnings

## Computational Social Science

# Computational Social Science

“It is an approach to social inquiry defined by (1) the use of large, complex datasets, often—though not always— measured in terabytes or petabytes; (2) the frequent involvement of “naturally occurring” social and digital media sources and other electronic databases; (3) the use of computational or algorithmic solutions to generate patterns and inferences from these data; and (4) the applicability to social theory in a variety of domains from the study of mass opinion to public health, from examinations of political events to social movements”

Shah, D. V., Cappella, J. N., & Neuman, W. R. (2015). Big Data, digital media, and computational social science: Possibilities and perils. *The ANNALS of the American Academy of Political and Social Science*, 659(1), 6–13.  
doi:10.1177/0002716215572084

# Computational Social Science

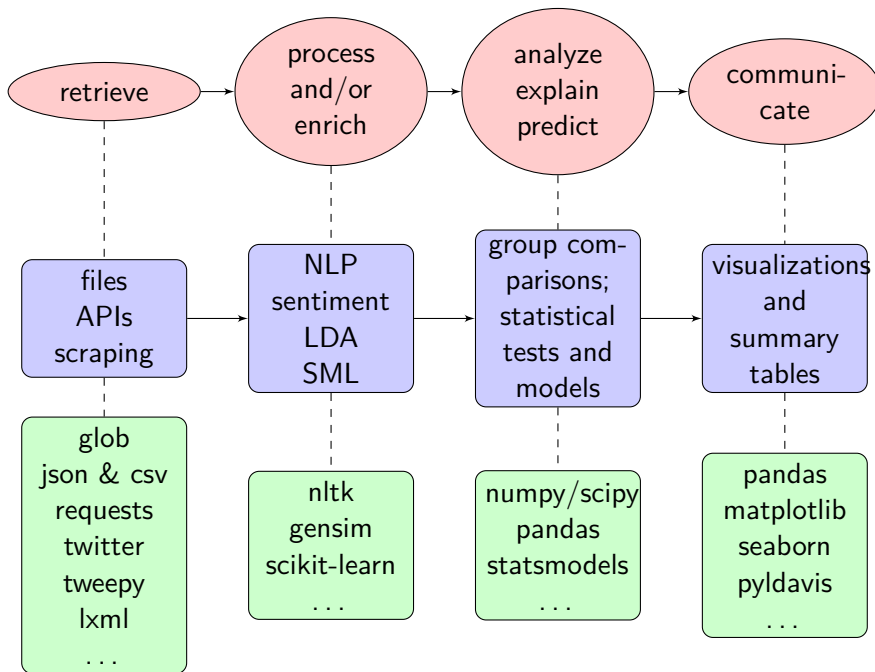
“[...] the computational social sciences employ the scientific method, complementing descriptive statistics with inferential statistics that seek to identify associations and causality. In other words, they are underpinned by an epistemology wherein the aim is to produce sophisticated statistical models that explain, simulate and predict human life.”

Kitchin, R. (2014). Big Data, new epistemologies and paradigm shifts. *Big Data & Society*, 1(1), 1–12.  
doi:10.1177/2053951714528481

# Steps of a CSS project

We learned techniques for:

- retrieving data
- processing data
- analyzing data
- visualising data





## A good workflow

# The big picture

## Start with pen and paper

### ① Draw the Big Picture

# The big picture

## Start with pen and paper

- 1 Draw the Big Picture
- 2 Then work out what components you need

# Develop components separately

## One script for downloading the data, one script for analyzing

- Avoids waste of resources (e.g., unnecessary downloading multiple times)

# Develop components separately

## One script for downloading the data, one script for analyzing

- Avoids waste of resources (e.g., unnecessary downloading multiple times)
- Makes it easier to re-use your code or apply it to other data

# Develop components separately

## One script for downloading the data, one script for analyzing

- Avoids waste of resources (e.g., unnecessary downloading multiple times)
- Makes it easier to re-use your code or apply it to other data

# Develop components separately

## One script for downloading the data, one script for analyzing

- Avoids waste of resources (e.g., unnecessary downloading multiple times)
- Makes it easier to re-use your code or apply it to other data

## Start small, then scale up

- Take your plan (see above) and solve *one* problem at a time (e.g., parsing a review page; or getting the URLs of all review pages)

# Develop components separately

## One script for downloading the data, one script for analyzing

- Avoids waste of resources (e.g., unnecessary downloading multiple times)
- Makes it easier to re-use your code or apply it to other data

## Start small, then scale up

- Take your plan (see above) and solve *one* problem at a time (e.g., parsing a review page; or getting the URLs of all review pages)
- (for instance, by using functions [next slides])



# Develop components separately

If you copy-paste code, you are doing something wrong

- Write loops!

# Develop components separately

If you copy-paste code, you are doing something wrong

- Write loops!
- If something takes more than a couple of lines, write a function!

## Copy-paste approach (ugly, error-prone, hard to scale up)

```
1 allreviews = []
2
3 response = requests.get('http://xxxxx')
4 tree = fromstring(response.text)
5 reviewelements = tree.xpath('//div[@class="review"]')
6 reviews = [e.text for e in reviewelements]
7 allreviews.extend(reviews)
8
9 response = requests.get('http://yyyyy')
10 tree = fromstring(response.text)
11 reviewelements = tree.xpath('//div[@class="review"]')
12 reviews = [e.text for e in reviewelements]
13 allreviews.extend(reviews)
```

Better: for-loop

(easier to read, less error-prone, easier to scale up (e.g., more URLs, read URLs from a file or existing list)))

```
1 allreviews = []
2
3 urls = ['http://xxxxx', 'http://yyyyy']
4
5 for url in urls:
6     response = requests.get(url)
7     tree = fromstring(response.text)
8     reviewelements = tree.xpath('//div[@class="review"]')
9     reviews = [e.text for e in reviewelements]
10    allreviews.extend(reviews)
```

Even better: for-loop with functions  
(main loop is easier to read, function can be re-used in multiple contexts)

```
1 def getreviews(url):
2     response = requests.get(url)
3     tree = fromstring(response.text)
4     reviewelements = tree.xpath('//div[@class="review"]')
5     return [e.text for e in reviewelements]
6
7
8 urls = ['http://xxxxx', 'http://yyyyy']
9
10 allreviews = []
11
12 for url in urls:
13     allreviews.extend(getreviews(url))
```

And you can always do even better: including a docstring

```
1 def getreviews(url):
2     '''scrapes all reviews from a given URL and returns a list of
3         strings'''
4     response = requests.get(url)
5     tree = fromstring(response.text)
6     reviewelements = tree.xpath('//div[@class="review"]')
7     return [e.text for e in reviewelements]
8
9 urls = ['http://xxxxx', 'http://yyyyy']
10
11 allreviews = []
12
13 for url in urls:
14     allreviews.extend(getreviews(url))
```

# Scaling up

If you continue working in this field, look into aspects like code style, re-usability, scalability

- Use functions and classes (Appendix D.3) to make code more readable and re-usable
- Avoid re-calculating values
- Think about how to minimize memory usage (e.g., Generators, Appendix D.2)
- Do not hard-code values, file names, etc., but take them as arguments

# Make it robust

You cannot foresee every possible problem.

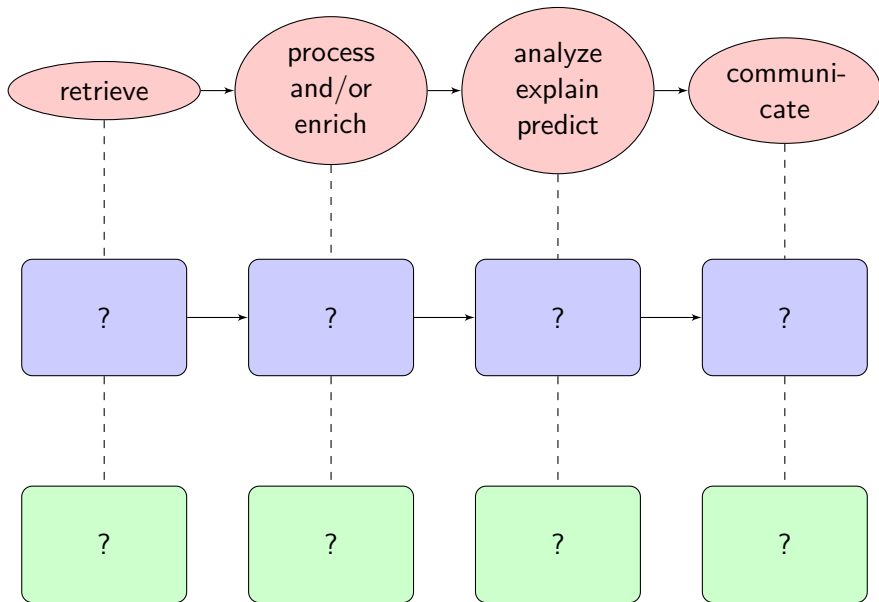
Most important: Make sure your program does not fail and loose all data just because something goes wrong at case 997/1000.

- Use try/except to explicitly tell the program how to handle errors
- Write data to files (or database) in between
- Use `assert len(x) == len(y)` for sanity checks



Looking forward

**What other possibilities do exist for each step?**



# Retrieve

## Webscraping with Selenium

- If content is dynamically loaded (e.g., with JavaScript), our approach doesn't work (because we don't have a browser).
- Solution: Have Python literally open a browser and literally click on things
- ⇒ Appendix E

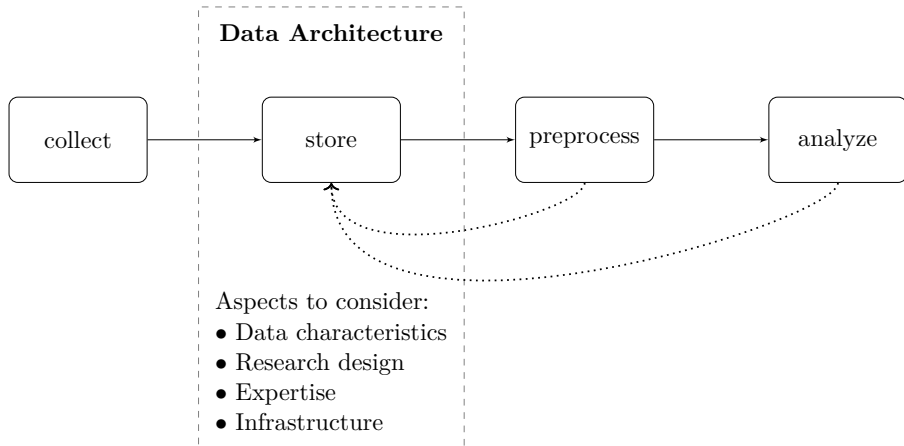
# Retrieve

## Use of databases

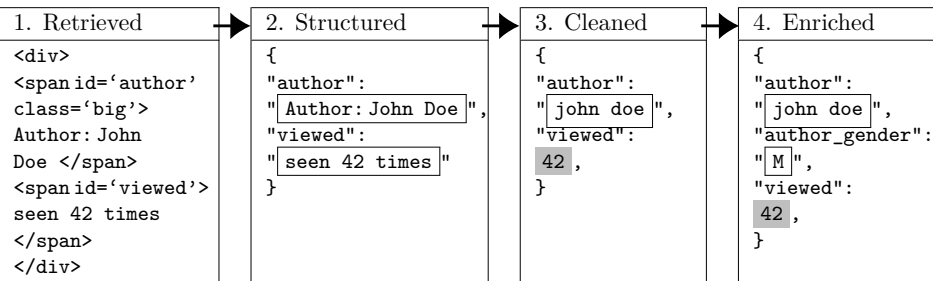
We did not discuss how to actually store the data

- We basically stored our data in files (often, one CSV or JSON file)
- But that's not very efficient if we have large datasets; especially if we want to select subsets later on
- SQL-databases to store tables (e.g., MySQL)
- NoSQL-databases to store less structured data (e.g., JSON with unknown keys) (e.g., MongoDB, Elasticsearch)
- ⇒ Günther, E., Trilling, D., & Van de Velde, R.N. (2018). But how do we store it? (Big) data architecture in the social-scientific research process. In: *Stuetzer, C.M., Welker, M., & Egger, M. (eds.): Computational Social Science in the Age of Big Data. Concepts, Methodologies, Tools, and Applications*. Cologne, Germany: Herbert von Halem.

# Storing data



# From retrieved data to enriched data



# Process and/or enrich

## Advanced NLP

We did a lot of BOW (and some POS-tagging), but we can get more

- Named Entity Recognition (NER) to get names of people, organizations, ...
- Dependency Parsing to find out exact relationships  $\Rightarrow$  nltk, Stanford, FROG, spacy. And now (that one is really cool): spacy

# Process and/or enrich

## Use images

- Supervised Machine learning does not care about what the features mean, so instead of texts we can also classify pictures
- Example: Teach the computer to decide whether an avatar on a social medium is an actual photograph of a person or a drawn image of something else
- This principle can be applied to many fields and disciplines – for example, it is possible to teach a computer to indicate if a tumor is present or not on X-rays of people's brains



# Process and/or enrich

## Use images

- To learn more about this, the following websites useful information: <http://blog.yhat.com/posts/image-classification-in-Python.html> and <http://cs231n.github.io/python-numpy-tutorial/#numpy-arrays>
- Possible workflow: Pixel color values as features  $\Rightarrow$  PCA to reduce features  $\Rightarrow$  train classifier
- Advanced stuff: Neural Networks

# Analyze/explain/predict

## More advanced modelling

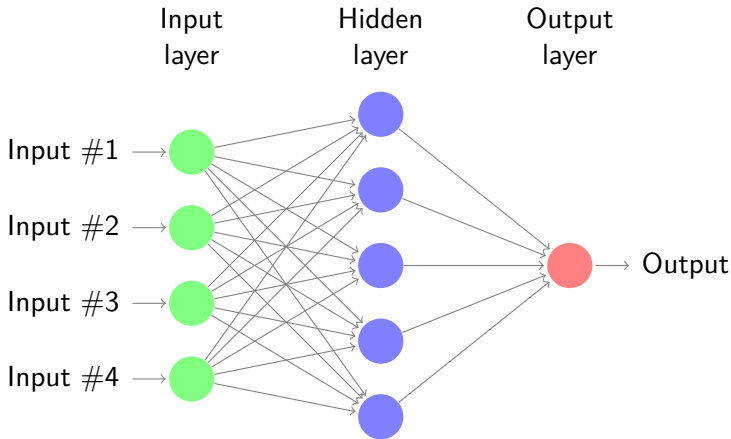
We only did some basic statistical tests

- There are more advanced regression techniques and dimension-reduction techniques tailored to data that are, e.g., large-scale, sparse, have a lot of features, ...
- $\Rightarrow$  scikit-learn, statsmodels

# Neural Networks and Deep Learning

# Neural Networks

- In “classical” machine learning, we predict an outcome directly based on the input features
- In neural networks, we can have “hidden layers” that we predict
- These layers are not necessarily interpretable
- “Neurons” that “fire” based on an “activation function”



⇒ If we had multiple hidden layers in a row, we'd call it a *deep* network.

# Why neural networks?

- learn hidden structures (e.g., embeddings (!))
- go beyond the idea that there is a direct relationship between occurrence of word X and label (or occurrence of pixel [R,G,B] and a label)
- images, machine translation — and more and more general NLP, sentiment analysis, etc.
- many Python frameworks available (e.g., Keras)

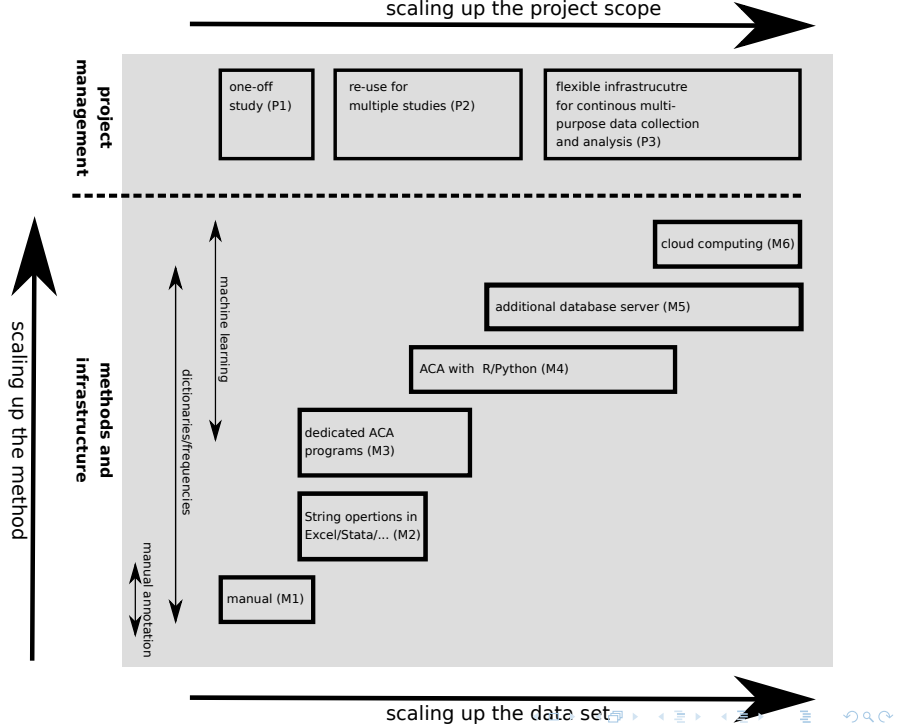
Example of a comparatively easy introduction:

<https://towardsdatascience.com/>

neural-network-embeddings-explained-4d028e6f0526

An example for scaling up:  
**The INCA project**

see also Trilling, D., & Jonkman, J.G.F. (2018). Scaling up content analysis. *Communication Methods and Measures*,  
doi:10.1080/19312458.2018.1447655





# INCA

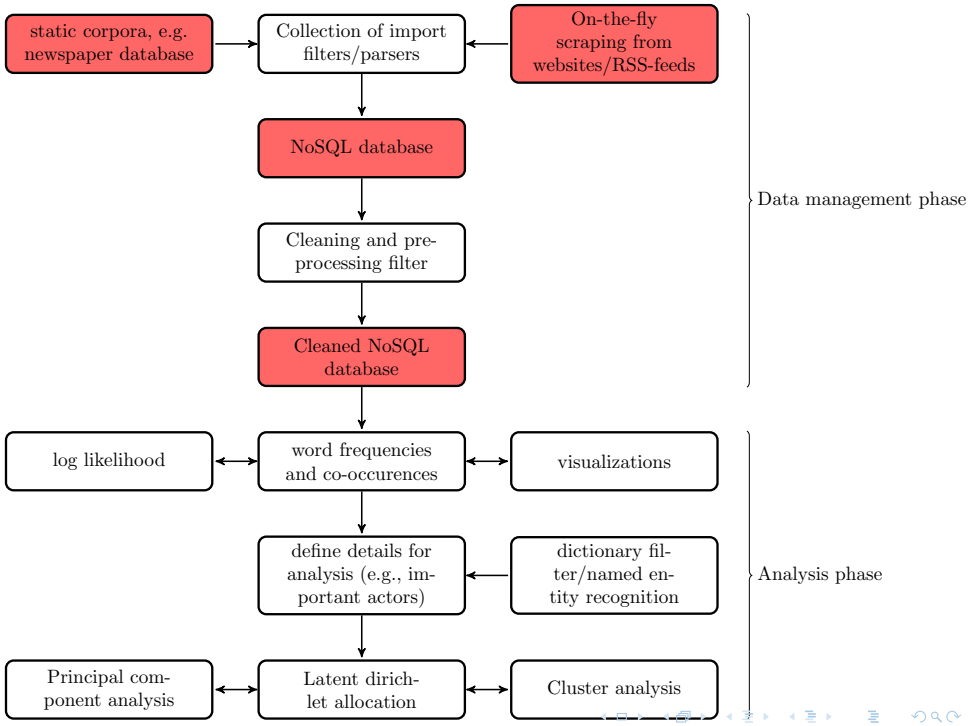
## How do we move beyond one-off projects?

- Collect data in such a way that it can be used for multiple projects
- Database backend
- Re-usability of preprocessing and analysis code

# INCA

## The idea

- Usable with minimal Python knowledge
- The “hard stuff” is already solved: writing a scraper often only involves replacing the XPATHs



INCA

and if you want to:

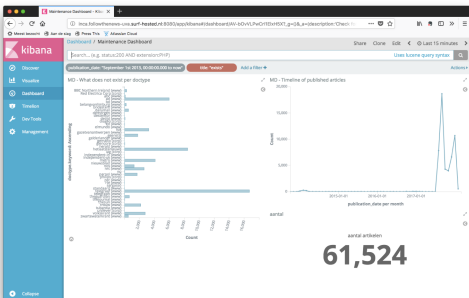
- export at each stage
- complex search queries

Scrape

Process

Analyze

ElasticSearch database



## analytics dashboard (KIBANA)

**Direct access to all  
functionality via  
Python**

```
g = inca.analysis.timeline_analysis.timeline_generator()
g.analyse('moslim', "publication_date", granularity='year', from_time='2014')

timestamp                                moslim
0  2011-01-01T00:00:00.000Z                1631
1  2012-01-01T00:00:00.000Z                1351
2  2013-01-01T00:00:00.000Z                1221
3  2014-01-01T00:00:00.000Z                2333
4  2015-01-01T00:00:00.000Z                2892
5  2016-01-01T00:00:00.000Z                2253
6  2017-01-01T00:00:00.000Z                2680
```

# Next meeting

## Deadline final exam

Hand in via filesender with your lab session teacher.

One .zip or .tar.gz file with

- .py and/or .ipynb for code (if multiple files: additional README.txt to explain what needs to be run when and why)
- .pdf for text and figures
- .csv, .json, or .txt for data
- any additional file I need to understand or reproduce your work

Good luck!!!