Computational Communication Science 2
Week 8 - Lecture
»Coding in an academic context«

Marthe Möller

a.m.moller@uva.nl

May 22, 2023

Digital Society Minor, University of Amsterdam

Today

Recap

A critical reflection

Open Science

About Reproducible Code

Looking Back and Ahead

Recap

Recap

Techniques you now master:

- Text as data
- Recommender systems
- Supervised machine learning

Today, we:

- Reflect on computational methods
- Discuss responsible coding

A critical reflection

Relevance of Computational Techniques

What can we use Python for?

- Methodological advancement: Using Python in traditional methods.
  - Building a recommender system and using it in an experiment

Relevance of Computational Techniques

What can we use Python for?

- Analzying text as a goal: Studying text can teach us a lot about human behavior.
  - Studying what people discuss on cancer-related online platforms (e.g., Sanders et al., 2020)

Relevance of Computational Techniques

What can we use Python for?

- Analyzing text as a means: Studying text can answer broader questions.
  - Auomatically distinguish between reliable and unreliable online information about vaccines by investigating what characterizes reliable and unreliable texts (e.g., Meppelink et al., 2021)

Three gaps in the development of CTAM

Baden et al. (2022):

- Measurement validity versus technological properties.

Three gaps in the development of CTAM

Baden et al. (2022):

- Measurement validity versus technological properties.
- Focus on detecting single and simple constructs instead of multiple and complex ones.

Three gaps in the development of CTAM

Baden et al. (2022):

- Measurement validity versus technological properties.
- Focus on detecting single and simple constructs instead of multiple and complex ones.
- Methods to study non-English texts underveloped.

Consequences for scholars

Baden et al. (2022):

- CTAM is applied to varying degrees in various disciplines.
- Blurring of theoretical concepts.

Consequences for society

The imperfections of computational methods also have broader consequences.

The harms of computing

Bender et al. (2021):

- CO2 emissions associated with training and developing models.
- Cost of and access to hardware.

"It is past time for researchers to prioritize energy efficiency and cost to reduce negative environmental impact and inequitable access to resources — both of which disproportionately affect people who are already in marginalized positions." (Bender et al., 2022, p. 613)

About that input (again)

Size does not garantuee diversity:

- Internet access is not evenly distributed
- Moderation causes marginalized populations to be less welcomed.
- Changes in social views cannot be accounted for when retraining models is expensive.

Bender et al., 2022

Potential solutions

Baden et al. (2022):

- Focus on validation.
- Combining strategies.
- Scholars as method developers instead of users.
- Define and operationalize constructs.
- Explain and argue for all the (pre-processing) that you do.
- Make the materials that you create publicly available.

Potential solutions

Sustainable computing

Bender et al. (2021):

- Report training time.
- Add efficiency as an evaluation metrics.
- Reflect on who your models are valuable for.
- Invest in better training data.

Potential solutions

This all starts with us!

Open Science

Open Computational Science

Sharing materials - advantages:

- Contributes to solving the problems outlined by scholars (e.g., Baden et al., 2022).
- Increases the transparency of your own projects.
- Increases the availability of and access to materials.
- Reduces the need to keep reinventing the wheel.

Open Computational Science

Sharing materials - concerns:

- Privacy of content creators (Hirschberg & Manning, 2015).
- Who owns these data? Consider the role of platforms.
  - Never make data publicily availble without ethical permission from the university!
- Who owns the code? You! But:
  - You do not own any packages or modules that you used, so you need to give proper credit.

About Reproducible Code

Reproducible code

Reproducible code is:

- Clear
- Readible
- Efficient

Reproducible code

PEP: Python Enhancement Proposal

Writing code efficiently

Pep8: APA for Python code

Goal: To improve the readibility of code.

See: https://peps.python.org/pep-0008

The Zen of Python (Peters, 2004): PEP20

Runs with command "import this"

Reproducible code

A selection of The Zen of Python (Peters, 2004):

Simple is better than complex.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Errors should never pass silently. Unless explicitly silenced.

Read all 19 rules on: https://peps.python.org/pep-0020/

Reproducible code

How to achieve reproducible code?

Open Computational Science

1. Document your code

Open Computational Science

Documentation debt: "When we rely on ever larger datasets we risk incurring documentation debt, i.e. putting ourselves in a situation where the datasets are both undocumented and too large to document post hoc." (Bender et al., p. 615)

Source Acknowledgement

```
"""
In your code, you can use """ or # to create a comment explaining your
    code. For example, to say that you are using a scraper developed by
     A. Person (2015), which can be found on www.somewebsite.com
"""
```

Writing reproducible code

2. Create functions (instead of repeating code)

Not:

```
# Rerun this code three times, once for each name:
# Mike, Elsa, and Minna

print("[change name here] is cool!")
```

But:

```
names = ["Mike", "Elsa", "Minna"]

def cool_caller(name):
  print(name + " is cool!")

for name in names:
  cool_caller(name)
```

Writing reproducible code

3. Eliminate unnecessary operations and break loops

Only interested in the first five elements of a list? Don't loop over the entire list!

Writing reproducible code

4. Avoid hard-coding values

Not: "myfile.csv" or 50 within your script.

But:

```
OUTPUTFILE ="myfile.csv"
MAXNUMBER=50
```

Writing reproducible code

5. Avoid defining unnecessary variables

Writing reproducible code

6. Use packages

And load all modules/packages at the start of your code

Writing reproducible code

7. Use informative names for files and variables (check out PEP8!)

Writing reproducible code

8. Keep lines of code fairly short

Writing reproducible code

9. Position assignments close to their usage.

Writing reproducible code

10. Write code that makes sense to others, and it will make snese to the future you as well.

Looking Back and Ahead

Looking back

You started with the basics (e.g., what is a list, how to save data, write a loop).

In two months, you learned:

- How to read in data
- How to preprocess data
- How transform text into data that a computer can understand
- How to compare texts to provide a recommendation
- How to analyze text to classify it automatically

Looking at the very near future

Two grades for this course remain:

- The last MC-questions
- The take-home exam

Looking at the near future

Final course of the minor: the research project!

- You will combine your programming skills with your skills as a researcher
- Run a research project about ComScience using the materials you created for the group assignment in this course
- More information follows in the first meeting of the research project

Looking at the future

CCS-1 and CCS-2: An introduction to coding. You can continue to learn and work with Python.

No course materials and instructors to help you out, but there are a lot of resources online!

Our tips:

- Error message? Google is your best friend!
- Check out the documentation of any module that you use to learn how it works
- Check out pubs using the method you are interested in. Often, they publish their python scripts (e.g., Meppelink et al., 2021)

Looking at the future

When you use your computational skills for your thesis or any other project, you are part of the research community. You add value to your own work by making it accessible and understandable for others!

Looking at the future

We taught you basic principles and techniques that underly frequently used methods. These techniques develop and change constantly - make sure that you stay updated on them!

Looking at the future

We taught you how to drive, now you can go out and explore the world of computational (communication) science!

Looking at the future

Thank you for the past weeks and enjoy the research project!