Computational Communication Science 2
Week 1 - Online Lecture
»Introduction & Lift Off«

Anne Kroon
Marthe Möller
a.c.kroon@uva.nl, @annekroon
a.m.moller@uva.nl, @marthemoller

April 1, 2024

Digital Society Minor, University of Amsterdam

Today

Introducing. . . the people

Introducing. . . the course

Text as data

The toolkit

- Bottom-up vs. top-down

Preprocessing

From test to large-scale

The people

Introducing. . . Marthe

dr. A. Marthe Möller

Assistant Professor Entertainment Communication

- Studying entertainment experiences in the digital space using:
  - Computational methods (e.g., ACA of user comments)
  - Experimental methods

@marthemoller |a.m.moller@uva.nl |https://www.uva.nl/profiel/m/o/a.m.moller/a.m.moller.html

Introducing...Anne

dr. Anne Kroon

Associate Professor Corporate Communication

- Research focus on biased AI in recruitment, and media bias regarding minorities
- Text analysis using automated approaches, word embeddings

@annekroon |a.c.kroon@uva.nl |
http://www.uva.nl/profiel/k/r/a.c.kroon/a.c.kroon.html

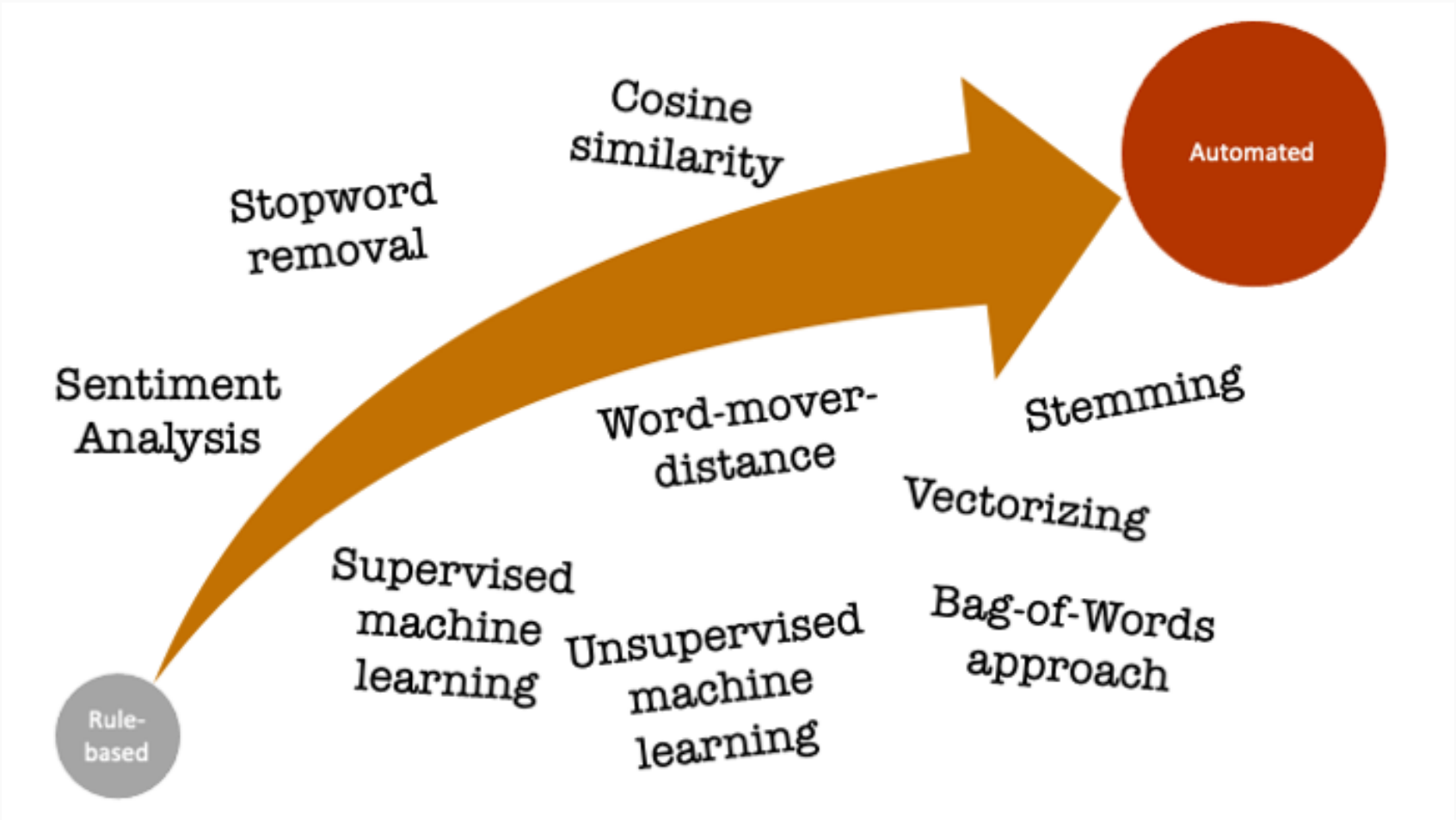
The course

About CCS-2

What is CCS-2?

- Next step after CCS-1
- Learn how to use what you learned in CCS-1 for research
- Expand on what you learned in CCS-1
  - Learn computational techniques (e.g. data vectorization, machine learning)
  - Learn how to use these techniques for research (e.g., automated content analysis)
- By the end of the course, you'll be prepared for the *Research Project*

About CCS-2


Automated
Cosine similarity
Stopword removal
Sentiment Analysis
Word-mover-distance
Stemming
Vectorizing
Supervised machine learning
Unsupervised machine learning
Bag-of-Words approach
Rule-based

About CCS-2

What will we do in this course?

- We discuss techniques in the lectures (Mondays)
- We practice with techniques in the tutorials (Tuesdays)
- Graded assignments to master the techniques:
  - Regular multiple choice questions (20%) about the readings and techniques that we discuss
    - 4 questions, in week 2, 3, 4, 6, 7
    - *total of 20 questions. 16 correct answers = full marks*
  - Group assignment: Get more experienced with the techniques and build a *basic* recommender system
    - Written report and code assignment (30%)
  - Final assignment (50%): at the end of the course so you can show off what you learned
- We provide structure through the meetings and assignments, you do the (home-)work

About CCS-2

Course set-up:

- *Lectures*: Introduction to techniques used in computational communication science
- *Tutorial meetings*: Lab sessions to learn how to work with these techniques
    - Possibility to ask questions about your code

All course materials can be found at. . .

https://github.com/uva-cw-ccs2/2324s2

About CCS-2

How to stay informed and where to find all the materials? Regularly check:

- The course Canvas page
- Your email
- The course Github page

In addition, make sure that you read the course manual so that you know all the ins and outs of this course!

Ready? Set? Go!

Without further ado. . .

. . . let's get started!

Text as data

Text as Data

CCS-1: You learned how to...

- Work with Python, for example, you:
  - Store text in json-files, csv-files etc.
  - Difference between a dict, a list, a string etc.
  - Work with data (e.g., creating a loop)

Text as Data: Analyzing text as a goal

Studying text can teach us a lot about human behavior:

- . . . to study the content cancer-related online platforms (e.g., Sanders et al., 2020)
  - what *topics* are being discussed on expert and peer-generated platforms?
- . . . does content differ between online and print news? (e.g., Burggraaff and Trilling, 2020)
  - E.g., online, journalists are more likely to publish *follow-up* articles.

Text as Data: Analyzing text as a means

Studying text can give us information we can use to answer broader questions:

- . . . analyze textual information about movies from IMDB to learn about the representation of women in movies (e.g.,Poma-Murialdo, 2019)
- . . . automatically distinguish between reliable and unreliable online information about vaccines by investigating what characterizes reliable and unreliable texts (e.g.,Meppelink et al., 2021)

Text as data: NLP

Natural language processing (NLP) refers to the branch of computer science — and more specifically, the branch of artificial intelligence or AI — concerned with giving computers the ability to understand text and spoken words in much the same way human beings can."
(IBM, 2020)

Watch this Ted talk for some inspiration if you like: https://www.ted.com/talks/jean_baptiste_michel_erez_lieberman_aiden_what_we_learned_from_5_million_books

The toolkit

The toolkit

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 CCS toolbox

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

A simple bottom-up approach

```
1 from collections import Counter
2 texts = ["Communication in the Digital Society is a very very complex
  ↪  phenomenon", "I like to study it"]
3 bottom_up = []
4 for t in texts:
5     bottom_up.append(Counter(t.lower().split()).most_common(3))
6 print(bottom_up)
```

This results in:

```
1 [('very', 2), ('Communication', 1), ('in', 1)]
2 [('I', 1), ('like', 1), ('to', 1)]
```

Please note that you can also write this like:

```
1 bottom_up = [Counter(t.split()).most_common(3) for t  in texts]
```

- This is *exactly* the same, just shorter (and faster).
- You do *not* have to use list comprehensions, but it helps if you can read them.

A simple top-down approach

```
texts = ["Communication in the Digital Society is a very very complex phenomenon", "I like to study it"]
features = ["communication", "digital", "study"]
for t in texts:
    print(f"\nAnalyzing '{t}':")
        for f in features:
            print(f"{f} occurs {t.lower().count(f)} times")
```

```
Analyzing 'Communication in the Digital Society is a very very complex phenomenon':
communication occurs 1 times
digital occurs 1 times
study occurs 0 times

Analyzing 'I like to study it':
communication occurs 0 times
digital occurs 0 times
study occurs 1 times
```

. . . save the results as a `list` as follows . . .

```
top_down = [[t.lower().count(f) for f in features] for t in texts]
```

When would you use which approach?

Some considerations

- Both can have a place in your workflow (e.g., bottom-up as first exploratory step)
- You have a clear theoretical expectation? Bottom-up makes little sense.
- But in any case: you need to transform your text into something "countable".

Preprocessing

Preprocessing in NLP

- Text preprocessing in *Natural Language Processing*.
- Typical step to get textual data into a more structured format for subsequent analyses
- These steps will come back in the upcoming weeks when we discuss bottom-up and top-down techniques

Typical preprocessing steps

Preprocessing steps

tokenization How do we (best) split a sentence into tokens (terms, words)?

pruning How can we remove unneccessary words/ punctuation?

lemmatization and stemming How can we make sure that slight variations of the same word are not counted differently?

ngrams Neighbouring terms

Simple string methods

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!

OK, good enough, perfect?

.split()

- space → new word
- no further processing whatsoever
- thus, only works well if we do a preprocessing outselves (e.g., remove punctuation)

```
1 docs = ["This is a text",  "I haven't seen John's derring-do. Second
  ↪  sentence!"]
2 tokens = [d.split() for d in docs]
```

```
1 [['This', 'is', 'a', 'text'], ['I', "haven't", 'seen', "John's",
2 'derring-do.', 'Second', 'sentence!']]
```

OK, good enough, perfect?

Tokenizers from the NLTK package

- multiple improved tokenizers that can be used instead of .split()
- e.g., Treebank tokenizer:
  - split standard contractions ("don't")
  - deals with punctuation

```
1 from nltk.tokenize import TreebankWordTokenizer
2 tokens = [TreebankWordTokenizer().tokenize(d) for d in docs]
```

```
1 [['This', 'is', 'a', 'text'], ['I', 'have', "n't", 'seen', 'John',
↪  "'s", 'derring-do.', 'Second', 'sentence', '!']]
```

Notice the failure to split the . at the end of the first sentence in the second doc. That's because `TreebankWordTokenizer` expects *sentences* as input. See book for a solution.

Stopword removal

- *The logic of the algorithm is very much related to the one of a simple sentiment analysis!*

Stopword removal

What are stopwords?

- Very frequent words with little inherent meaning
- `the, a, he, she, ...`
- context-dependent: if you are interested in gender, `he` and `she` are no stopwords.
- Many existing lists as basis

Stopword removal: What and why?

Why remove stopwords?

- If we want to identify key terms (e.g., by means of a word count), we are not interested in them
- If we want to calculate document similarity, it might be inflated
- If we want to make a word co-occurance graph, irrelevant information will dominate the picture

Stopword removal

```
1  from nltk.corpus import stopwords
2  mystopwords = stopwords.words("english")
3  mystopwords.extend(["test", "this"])
4
5  tokens_without_stopwords = [[word for word in doc if word.lower() not
   ↪  in mystopwords] for doc in tokens]
```

```
1  [['text'], ["n't", 'seen', 'John', 'derring-do.', 'Second',
   ↪  'sentence', '!']]
```

You can do more!

For instance, you could add an `or` statement to also exclude punctuation.

Removing punctuation

```
1 from nltk.tokenize import RegexpTokenizer
2 tokenizer = RegexpTokenizer(r'\w+')
3 tokenizer.tokenize("Hi students, what's up!")
```

```
1 ['Hi', 'students', 'what', 's', 'up']
```

Stemming and lemmatization

Why do we need this?

- Because we do not want to distinguish between smoke, smoked, smoking, ...
- Typical preprocessing step (like stopword removal)
- Ultimate aim: normalize text

Stemming

- Stemming: reduce words to its stem by removing last part (drinking $\rightarrow$ drink)
- Lemmatization: find word that you would need to look up in a dictionary (drinking $\rightarrow$ drink, but also went $\rightarrow$ go)
- stemming is simpler than lemmatization
- lemmatization often better

```python
import nltk
import spacy
porter_stemmer = nltk.stem.PorterStemmer() # NLTK's Porter Stemmer
nlp = spacy.load("en_core_web_sm")# Load spaCy's English language model
sentence = "The dogs are barking, the cats are playing, the girl sang a song"
# Use NLTK's Porter Stemmer to stem the sentence
stemmed_sentence = " ".join([porter_stemmer.stem(word) for word in sentence.split()])
# Use spaCy's English language model to lemmatize the sentence
lemmatized_sentence = " ".join([token.lemma_ for token in nlp(sentence)])
# Print the original sentence, stemmed sentence, and lemmatized sentence
print(sentence)
print(stemmed_sentence)
print(lemmatized_sentence)
```

```
The dogs are barking, the cats are playing, the girl sang a song
the dog are barking, the cat are playing, the girl sang a song
the dog be bark , the cat be play , the girl sing a song
```

ngrams

Instead of just looking at single words (unigrams), we can also use adjacent words (bigrams).

ngrams

```
1 import nltk
2 texts = ['This is the first text text text first', 'And another text
  ↪  yeah yeah']
3 texts_bigrams = [["_".join(tup) for tup in nltk.ngrams(t.split(),2)]
  ↪  for t in texts]
4 print(texts_bigrams)
```

```
[['This_is', 'is_the', 'the_first', 'first_text',
'text_text', 'text_text', 'text_first'],
['And_another', 'another_text', 'text_yeah',
'yeah_yeah']]
```

Typically, we would combine both uni and bigrams.

What do you think? Why is this useful? (and what may be drawbacks?)

Main takeaway

- Preprocessing matters, be able to make informed choices.
- Keep this in mind when moving to Machine Learning.

From test to large-scale

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 ow= 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.

Tuesday April 2

Tutorial meeting tomorrow

- Start to practice with preprocessing techniques yourself
- Try the code in these slides at home. Make sure you can follow along.
- Use the tutorial meetings to discuss your questions.

Practice is key!

Thank you!!

Thank you for your attention!

- Questions? Comments?

References i

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


Burggraaff, C., & Trilling, D. (2020). Through a different gate: An automated content analysis of how online news and print news differ. *Journalism*, *21*(1), 112–129. https://doi.org/10.1177/1464884917716699


Meppelink, C. S., Hendriks, H., Trilling, D., van Weert, J. C., Shao, A., & Smit, E. S. (2021). Reliable or not? an automated classification of webpages about early childhood vaccination using supervised machine learning. *Patient Education and Counseling*, *104*(6), 1460–1466. https://doi.org/10.1016/j.pec.2020.11.013


Poma-Murialdo, S. C. (2019). *Gender inequality in the movie industry:* (Master's Thesis). University of Amsterdam.

References ii

Sanders, R., Linn, A. J., Araujo, T. B., Vliegenthart, R., van Eenbergen, M. C., & van Weert, J. C. (2020). Different platforms for different patients' needs: Automatic content analysis of different online health information platforms. *International Journal of Human-Computer Studies*, *137*, 102386. https://doi.org/10.1016/j.ijhcs.2019.102386