Computational Communication Science 2
Week 1 - Lecture
»Introduction«

Marthe Möller
Anne Kroon

A.M.Moller@uva.nl, @MartheMoller
a.c.kroon@uva.nl, @annekroon

April, 2022

Digital Society Minor, University of Amsterdam

Today

Introducing...the people

Introducing...the course

Text as Data

Analyzing songtexts: NLP

All course materials can be found at...

https://github.com/annekroon/CCS-2

Introducing...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

Assistant 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

Introducing... You

Your name?
Your background?
Your reason for taking this course?
Do you have a dataset you are working on?

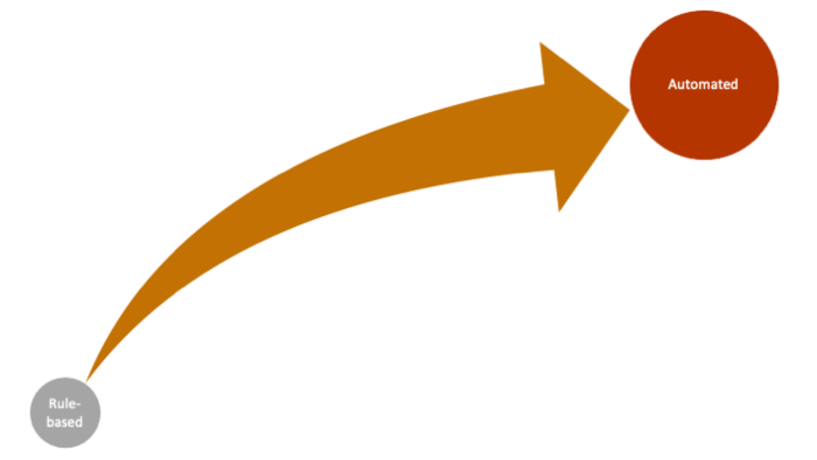
Introducing...the course

About CCS-2

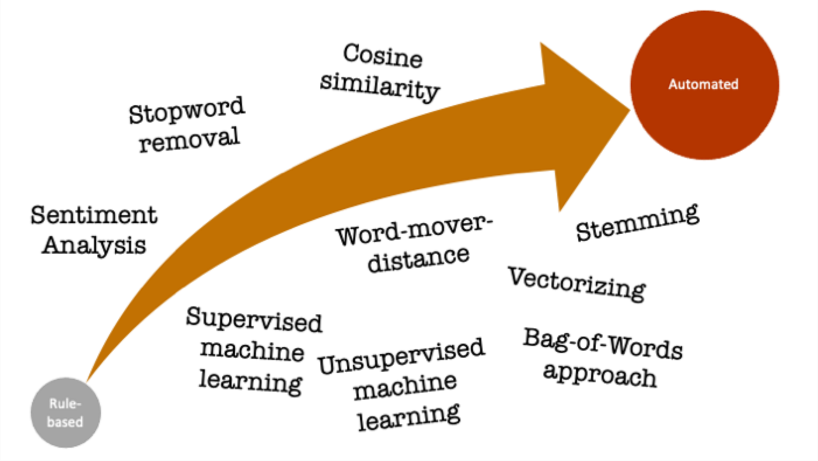
What is CCS-2?

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

About CCS-2



About CCS-2


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

About CCS-2

What will we do in this course?

- We discuss techniques in the lectures
- We practice with techniques in the tutorials
- Graded assignments to master the techniques:
  - Regular multiple choice questions (20%) about readings that use the techniques we discuss
  - Coding challenge (group assignment): Get more experienced with the techniques and build a recommender system
    - Report (20%)
    - Presentation (10%)
  - Take-home exam (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

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!

About CCS-2

How to contact Anne and Marthe?

We kunnen hier eventueel iets over zeggen: mogen ze mailen, zijn er spreekuren etc.?

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.
    - Work with texts in Python

Text as Data: Learning from text directly

Studying text can teach us a lot about human behavior:

What topics do people discuss on online cancer-related platforms?
(Sanders et al., 2020)

To what extent does content differ between online and print news?
(Burggraaff and Trilling, 2020)

What topics do people discuss in their movie reviews?
(Schneider et al., 2020)

Text as Data: Analzing 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
(Poma-Murialdo, 2019)

Automatically distinguish between reliable and unreliable online information about vaccines by investigating what characterizes reliable and unreliable texts
(Meppelink et al., 2021)

Text as Data: Combining text analysis with other methods

We can use data about text in combination with other methods:

Combining data about media content and survey data to investigate how media coverage affects citizens' trust in the EU (Brosius et al., 2019)

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)

Analyzing songtexts: NLP

Molly Malone

```
MollyMalone = "In Dublin's fair city, where the girls are so pretty, I first set my eyes on sweet Molly Malone. As she wheeled her wheelbarrow, Through streets broad and narrow Crying, Cockles and mussels, alive, alive, oh! Alive, alive, oh, Alive, alive, oh, Crying, Cockles and mussels, alive, alive, oh."
```

Molly Malone

```
print(type(MollyMalone))
print(len(MollyMalone))
print(MollyMalone[0])
print(MollyMalone[-1:])
```

```
<class 'str'>
96
I
.
```

NLTK

NLTK: Natural Language Toolkit (www.nltk.org)
Tokenization: The process of breaking text (paragraphs, sentences, etc.) into smaller parts (individual sentences, words, etc.)

Tokenization

```
MM_words = word_tokenize(MollyMalone)

print(MM_words)
```

```
['In', 'Dublin', "'s", 'fair', 'city', ',', 'where', 'the', '
    girls', 'are', 'so', 'pretty', ',', 'I', 'first', 'set', 'my
    ', 'eyes', 'on', 'sweet', 'Molly', 'Malone', '.', 'As', 'she
    ', 'wheeled', 'her', 'wheelbarrow', ',', 'Through', 'streets
    ', 'broad', 'and', 'narrow', 'Crying', ',', 'Cockles', 'and
    ', 'mussels', ',', 'alive', ',', 'alive', ',', 'oh', '!', '
    Alive', ',', 'alive', ',', 'oh', ',', 'Alive', ',', 'alive',
     ',', 'oh', ',', 'Crying', ',', 'Cockles', 'and', 'mussels',
     ',', 'alive', ',', 'alive', ',', 'oh', '.']
```

Tokenization

```
print(Counter(MM_words).most_common(3))
```

```
[(',', 17), ('alive', 6), ('oh', 4)]
```

Removing Stopwords

```
stopwords = ['in', 'the', 'and', 'a','I', 'she', 'her', 'are', 'so', 'on', 'me', 'my', 'mine', 'oh']
nostopwords = []

for word in MM_words:
   if word not in stopwords:
      nostopwords.append(word)

print(nostopwords)
```

```
['In', 'Dublin', "'s", 'fair', 'city', ',', 'where', 'girls', 'pretty', ',', 'first', 'set', 'eyes', 'sweet', 'Molly', 'Malone', '.', 'As', 'wheeled', 'wheelbarrow', ',', 'Through', 'streets', 'broad', 'narrow', 'Crying', ',', 'Cockles', 'mussels', ',', 'alive', ',', 'alive', ',', '!', 'Alive', ',', 'alive', ',', ',', 'Alive', ',', 'alive', ',', ',', 'Crying', ',', 'Cockles', 'mussels', ',', 'alive', ',', 'alive', ',', '.']
```

Removing Stopwords

```
print(Counter(nostopwords).most_common(3))
```

```
[(',', 17), ('alive', 6), ('.', 2)]
```

Removing Stopwords

```

from nltk.corpus import stopwords
print(len(stop_words))
```

```
179
```

Removing Stopwords

```python
nostopwords = []

for word in MM_words:
  if word not in stop_words:
    nostopwords.append(word)

print(nostopwords)
```

```
['In', 'Dublin', "'s", 'fair', 'city', ',', 'girls', 'pretty',
    ',', 'I', 'first', 'set', 'eyes', 'sweet', 'Molly', 'Malone
    ', '.', 'As', 'wheeled', 'wheelbarrow', ',', 'Through', '
    streets', 'broad', 'narrow', 'Crying', ',', 'Cockles', '
    mussels', ',', 'alive', ',', 'alive', ',', 'oh', '!', 'Alive
    ', ',', 'alive', ',', 'oh', ',', 'Alive', ',', 'alive', ',',
     'oh', ',', 'Crying', ',', 'Cockles', 'mussels', ',', 'alive
    ', ',', 'alive', ',', 'oh', '.']
```

Molly Malone

```python
print(Counter(nostopwords).most_common(3))
```

```
[(',', 17), ('alive', 6), ('oh', 4)]
```

Molly Malone

```python
import string
punct = list(string.punctuation)
print(punct[:5])
```

```
['!', '"', '#', '$', '%']
```

Molly Malone

```python
nostopnopunct = []

for word in nostopwords:
   if word not in punct:
      nostopnopunct.append(word)

print(nostopnopunct)
```

```
['In', 'Dublin', "'s", 'fair', 'city', 'where', 'girls', 'pretty', 'first', 'set', 'eyes', 'sweet', 'Molly', 'Malone', 'As', 'wheeled', 'wheelbarrow', 'Through', 'streets', 'broad', 'narrow', 'Crying', 'Cockles', 'mussels', 'alive', 'alive', 'Alive', 'alive', 'Alive', 'alive', 'Crying', 'Cockles', 'mussels', 'alive', 'alive']
```

Molly Malone

```
print(Counter(nostopnopunct).most_common(3))
```

```
[('alive', 6), ('Crying', 2), ('Cockles', 2)]
```

Molly Malone

```
lower = []

for word in nostopnopunct:
lower.append(word.lower())

print(lower)
```

```
['in', 'dublin', "'s", 'fair', 'city', 'where', 'girls', 'pretty
    ', 'first', 'set', 'eyes', 'sweet', 'molly', 'malone', 'as',
     'wheeled', 'wheelbarrow', 'through', 'streets', 'broad', '
    narrow', 'crying', 'cockles', 'mussels', 'alive', 'alive', '
    alive', 'alive', 'alive', 'alive', 'crying', 'cockles', '
    mussels', 'alive', 'alive']
```

Molly Malone

```
print(Counter(lower).most_common(3))
```

```
[('alive,', 8), ('crying', 2), ('cockles', 2)]
```

Stemming

```
words = (Counter(lower).most_common(3))

from nltk.stem.porter import *
stemmer = PorterStemmer()

stems = [stemmer.stem(words) for wrd in words]
print(stems)
^^I
```

```
out put here
^^I
```

Lemmatization

```
words = (Counter(lower).most_common(3))
^^I^^I
from nltk.stem import WordNetLemmatizer
lemmatizer = WorddNetLemmatizer()

lems = [lemmatizer.lemmatize(words) for wrd in words]
print(stems)
```

```
output here
```

Note

Liefst hier even een terugkoppeling naar een voorbeeld: wat kun jen u doen met lemmatization and stemming?

Analyzing songtexts: RegEx

RegEx

”A *regular expression* or *regex* is a powerful language to locate strings that conform to a given pattern. [...] Specifically, regular expressions are a sequence of characters that we can use to design a pattern and then use this pattern to *find* strings (identify or extract) and also *replace* those strings by new ones.”

Van Atteveldt et al., 2022

RegEx: Search, Match, or Findall

```
print(re.search("live", "Alive"))
```

```
live
```

```
print(re.match("live", "Alive"))
```

```
print(re.match("live", MollyMalone_words))
```

```
live
live
live
live
live
live
live
live
```

RegEx

```
for word in MollyMalone_words:
  if re.search("[Aa]live", word):
    print(word)
```

```
alive
alive
Alive
alive
Alive
alive
alive
alive
```

RegEx

`m[oa]lly` matches molly, but also mally

`m.lly` matches molly, but also mally, or melly, or milly...

Quantifiers

`mol+y` matches moly, molly, mollly, molllly...

`[mM]ol+y` matches moly, molly, and mollly, but also Moly, Molly, Mollly, Molllly...

Quantifiers

<b>Molly Malone</b>

`<.*>` is greedy and will select everything

`<.*?>` is non-greedy and will match <b> and </b>

Groups

`That was (not)? the end of sweet Molly Malone`

will select both:

That was the end of sweet Molly Malone and That was not the end of sweet Molly Malone

Character classes

The Dublin Millennium Commission proclaimed 13 June to be "Molly Malone Day"

```
[1-9]+.[A-Z][a-z]+
```

will select 13 June

note

En dan eindigen met: terug naar onderzoek, wat kun je hier nu precies mee? En dan noem je de Twitter-artikelen als voorbeeld, zodat er weer terugkoppeling is van pietje-precieze code dingen naar onderzoek

+ schietgebedje dat het niet allemaal veeeeel te veel is :)