

# 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

Analyzing songtexts: RegEx

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...the course

---

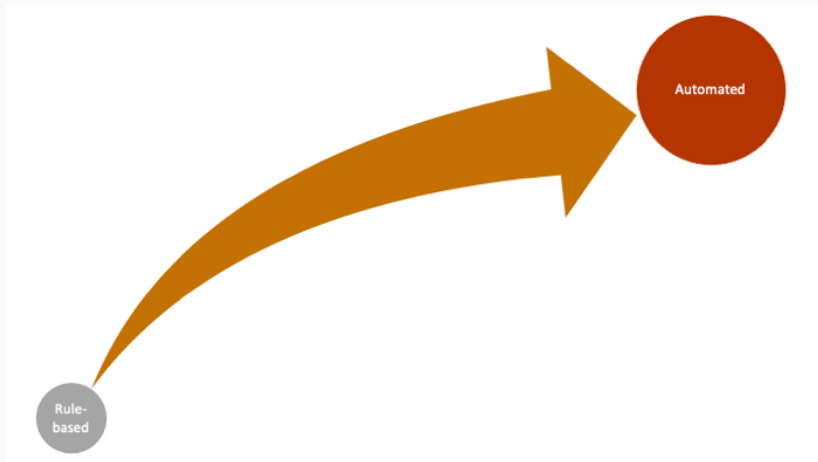
# About CCS-2

What is CCS-2?

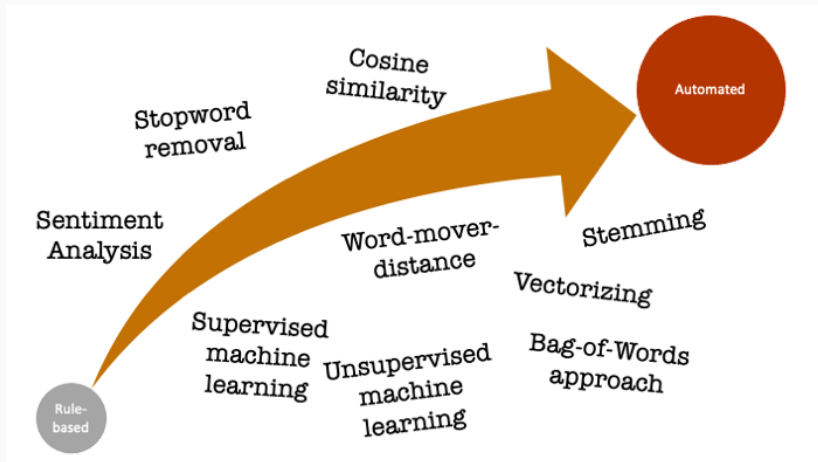
- Next step after CCS-1
- How to use what you learned in CCS-1 for research?
  - Learn computational techniques (e.g. data vectorization, machine learning)
  - Learn how to use these techniques for research (e.g., content analysis)
- By the end of the course, you'll be prepared to do computational research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



## About CCS-2



## About CCS-2



## 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 the readings and techniques that 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

### 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

## 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.
  - 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\_different\_2020)

To what extent does content differ between online and print news?  
(burggraaff\_through\_2020)

What topics do people discuss in their movie reviews?  
(schneider\_what\_2020)

## 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  
(poma-murialdo\_gender\_2019)

Automatically distinguish between reliable and unreliable online information about vaccines by investigating what characterizes reliable and unreliable texts  
(meppelink\_reliable\_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\_trust\_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

```
1 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

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

```
1 <class 'str'>
2 291
3 I
4 .
```

# NLTK

NLTK: Natural Language Toolkit ([www.nltk.org](http://www.nltk.org))

Tokenization: The process of breaking text (paragraphs, sentences, etc.) into smaller parts (individual sentences, words, etc.)



# Tokenization

```
1 MM_words = word_tokenize(MollyMalone)
2
3 print(MM_words)
```

```
1 ['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

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

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

# Removing Stopwords

```
1 stopwords = ['in', 'the', 'and', 'a', 'I', 'she', 'her', 'are', 'so', 'on',  
    'me', 'my', 'mine', 'oh']  
2 nostopwords = []  
3  
4 for word in MM_words:  
5     if word not in stopwords:  
6         nostopwords.append(word)  
7  
8 print(nostopwords)
```

```
1 ['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

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

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

# Removing Stopwords

```
1  
2 from nltk.corpus import stopwords  
3 stop_words = stopwords.words("english")  
4 print(len(stop_words))
```

```
1 179
```

# Removing Stopwords

```
1 nostopwords = []  
2  
3 for word in MM_words:  
4     if word not in stop_words:  
5         nostopwords.append(word)  
6  
7 print(nostopwords)
```

```
1 ['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

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

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

# Molly Malone

```
1 import string
2 punct = list(string.punctuation)
3 print(punct[:5])
```

```
1 ['!', '"', '#', '$', '%']
```



# Molly Malone

```
1 nostopnopunct = []
2
3 for word in nostopwords:
4     if word not in punct:
5         nostopnopunct.append(word)
6
7 print(nostopnopunct)
```

```
1 ['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

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

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

# Molly Malone

```
1 lower = []  
2  
3 for word in nostopnopunct:  
4     lower.append(word.lower())  
5  
6 print(lower)
```

```
1 ['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

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

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

## Zooming out

So far, we talked about:

- Natural Language Processing and the NLTK
- Tokenization

Next, we will talk about:

- Stemming and lemmatization

# Stemming

```
1 from nltk.stem.porter import *
2 stemmer = PorterStemmer()
3
4 stems = [stemmer.stem(word) for word in lower]
5 print(stems)
```

```
1 ['in', 'dublin', "'s", 'fair', 'citi', 'where', 'girl', 'pretti',
   'first', 'set', 'eye', 'sweet', 'molli', 'malon', 'as', '
   wheel', 'wheelbarrow', 'through', 'street', 'broad', 'narrow
   ', 'cri', 'cockl', 'mussel', 'aliv', 'aliv', 'aliv', 'aliv',
   'aliv', 'aliv', 'cri', 'cockl', 'mussel', 'aliv', 'aliv']
```

# Stemming

Compare the new data:

```
1 ['in', 'dublin', "'s", 'fair', 'citi', 'where', 'girl', 'pretti',  
  'first', 'set', 'eye', 'sweet', 'molli', 'malon', 'as', '  
  wheel', 'wheelbarrow', 'through', 'street', 'broad', 'narrow  
  ', 'cri', 'cockl', 'mussel', 'aliv', 'aliv', 'aliv', 'aliv',  
  'aliv', 'aliv', 'cri', 'cockl', 'mussel', 'aliv', 'aliv']
```

To the old data:

```
1 ['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']
```

# Lemmatization

```
1 from nltk.stem import WordNetLemmatizer
2 lemmatizer = WordNetLemmatizer()
3 nltk.download('wordnet')
4
5 lems = [lemmatizer.lemmatize(word) for word in lower]
6 print(lems)
```

```
1 ['in', 'dublin', "'s", 'fair', 'city', 'where', 'girl', 'pretty',
   'first', 'set', 'eye', 'sweet', 'molly', 'malone', 'a', '
   wheeled', 'wheelbarrow', 'through', 'street', 'broad', '
   narrow', 'cry', 'cockle', 'mussel', 'alive', 'alive', 'alive
   ', 'alive', 'alive', 'alive', 'cry', 'cockle', 'mussel', '
   alive', 'alive']
```



# Lemmatization

Compare lemmatization:

```
1 ['in', 'dublin', "'s", 'fair', 'city', 'where', 'girl', 'pretty',  
  'first', 'set', 'eye', 'sweet', 'molly', 'malone', 'a', '  
  wheeled', 'wheelbarrow', 'through', 'street', 'broad', '  
  narrow', 'cry', 'cockle', 'mussel', 'alive', 'alive', 'alive  
  ', 'alive', 'alive', 'alive', 'alive', 'cry', 'cockle', 'mussel', '  
  alive', 'alive']
```

To stemming:

```
1 ['in', 'dublin', "'s", 'fair', 'citi', 'where', 'girl', 'pretti',  
  'first', 'set', 'eye', 'sweet', 'molli', 'malon', 'as', '  
  wheel', 'wheelbarrow', 'through', 'street', 'broad', 'narrow  
  ', 'cri', 'cockl', 'mussel', 'aliv', 'aliv', 'aliv', 'aliv',  
  'aliv', 'aliv', 'cri', 'cockl', 'mussel', 'aliv', 'aliv']
```

## Zooming out

So far, we talked about:

- Natural Language Processing and the NLTK
- Tokenization
- Stemming and lemmatization

Next, we will talk about:

- Regular expressions

# 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\_computational\_2022

# RegEx: Search, Match, or Findall

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

```
1 live
```

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

re.match()

re.search()

re.findall()

# RegEx

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

```
1 alive
2 alive
3 Alive
4 alive
5 Alive
6 alive
7 alive
8 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]o1+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

## RegEx in ComScience

"Next to steps like conversion to lowercase and the removal of stop words and punctuation, we paid extensive attention to the recoding of potential firms mentioned in our data in order to circumvent underestimation of news attention to firms. We manually compiled a list of companies – based on existing lists of Dutch and Non-Dutch multinationals, and AEX stock market-listed firms potentially relevant for our study. If a firm on this list contained more than one word or had synonyms or different spellings, our program used regular expressions to replace these by a unique term that we defined. For example, AirFrance KLM and different variations on it were all converted into Air\_France\_KLM and ABN AMRO into ABN\_AMRO. In addition, we supplemented abbreviations like ABN by the full (new) name, if the full name was mentioned in the same text." Jonkman et al. (2016), p. 1615

## Zooming out

Today, we talked about:

- Natural Language Processing and the NLTK
- Tokenization
- Stemming and lemmatization
- Regular expressions

In this weeks' tutorial, you will:

- Start with the first MC-questions about the assigned reading for today and about the content of this lecture
- Practice with the processes discussed today, so that you can use them (amongst others) when you explore data next week!