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

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...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

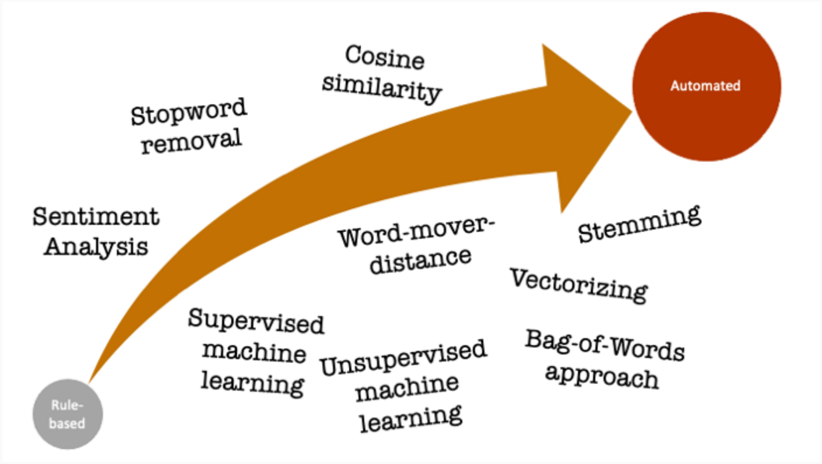
Introducing...the course

About CCS-2

What is CCS-2?

- Next step after CCS-1
- 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., content analysis)
- By the end of the course, you'll be prepared for the Research Project

About CCS-2


Cosine similarity
Stopword removal
Automated
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
- 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

All course materials can be found at...

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

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:

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

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

```
<class 'str'>
291
I
.
```

NLTK

What are the three most commonly used (meaningful) words in this song?

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

```python

import nltk
from nltk.tokenize import word_tokenize

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

```python
from collections import Counter

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

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

What about stopwords?

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)]
```

Stopword list provided by the NLTK.

Removing Stopwords

```python
from nltk.corpus import stopwords
stop_words = stopwords.words("english")
print(len(stop_words))
```

```
179
```

Removing Stopwords

```
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

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

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

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

```python
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)]
```

Many text analyses require steps ('preprocessing data') such as the ones discussed here.

Zooming out

So far, we talked about:

- Natural Language Processing and the NLTK
- Tokenization
- Preprocessing steps: stopword removal, punctuation removal, lowercase conversion

Next, we will talk about:

- Stemming and lemmatization

Zooming out

What if we want to compare many different songtexts to see how similar they are?

In such cases, it may help to "normalize" text so that different texts are better comparable.

Stemming

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

stems = [stemmer.stem(word) for word in lower]
print(stems)
```

```
['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:

```
['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:

```
['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

```
from nltk.stem import WordNetLemmatizer
lemmatizer = WorddNetLemmatizer()
nltk.download('wordnet')

lems = [lemmatizer.lemmatize(word) for word in lower]
print(lems)
```

```
['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:

```
['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']
```

To stemming:

```
['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
- Preprocessing steps: stopword removal, punctuation removal, lowercase conversion
- 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 et al., 2022

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. [...] our program used regular expressions to replace the different names of firms by a unique term that we defined."

Jonkman et al. (2016), p. 1615

Find the strings that conform to this pattern: ”live” .

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.

RegEx: Search, Match, or Findall

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

```
live
```

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

re.search()
re.match()
re.findall()

RegEx: I don't exactly know what I am looking for!

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

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

RegEx: Wildcards

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

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

RegEx: Quantifiers

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

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

By definition, quantifiers are greedy.

RegEx: Greedy vs. Lazy

<b>Molly Malone</b>

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

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

RegEx: 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]+\s[A-Z][a-z]+
```

will select 13 June

Character classes

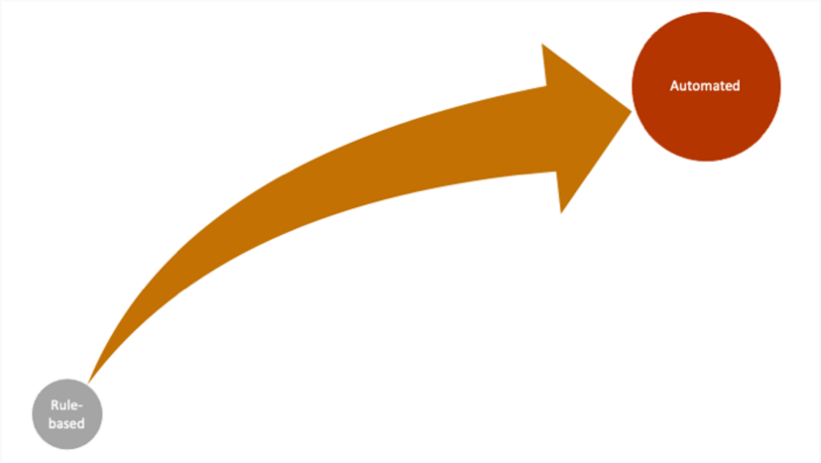
`[1-9]+\s[A-Z][a-z]+`

will select 13 June

but also 15 August

and 23 September

Back about this



Zooming out

Today, we talked about:

- Natural Language Processing and the NLTK
- Tokenization
- Preprocessing steps: stopword removal, punctuation removal, lowercase conversion
- 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!