Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

472867-HS2025-0: NLP and Text Mining

02 -Text Processing and SpaCy

murat.sariyar@bfh.ch

Video Resources

Suggestions:

https://www.youtube.com/watch?v=pph11STqtXQ

https://www.youtube.com/watch?v=z6wzF4nkpJM

https://www.youtube.com/watch?v=RhJgztc1g4Q

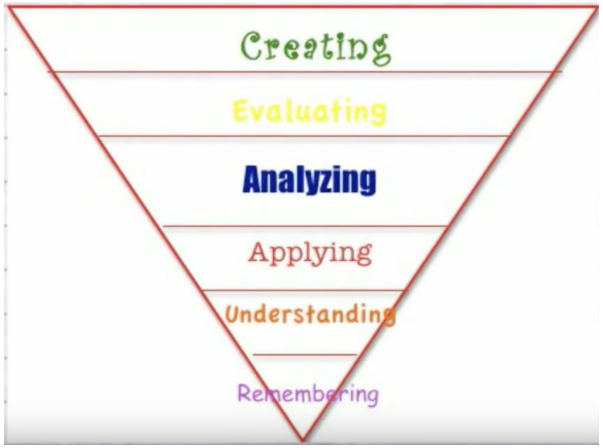
If you have time (one of the best videos, but 3h long):

https://www.youtube.com/watch?v=dIUTsFT2MeQ

Flipped Classroom

Learn the content on your own, at your own speed, using the course materials.

In class, bring your questions—we'll discuss homework and work through exercises together.



https://www.youtube.com/watch?v=iroPveV3dbk

Three Phases I

1) Before Class

- You receive the slides, homework, and links to resources.
- Read the slides and note your questions.
- Use the provided resources to attempt all homework tasks.
- Upload your solutions

Goal: Arrive prepared, with attempted solutions and a list of concrete questions.

2) In-Class Session

- We review outstanding questions together (using the Whiteboard).
- We discuss the homework and solve additional practice exercises collaboratively.

Goal: Clarify open issues and deepen understanding through live problem-solving.

Three Phases II

3) After Class

- One group is assigned to create a master document that consolidates:
  - the questions,
  - the tasks, and
  - the solutions.
- Use the template “Solutions-WeekX” in the “Solutions” folder.
- Upload the compiled document to Moodle.
- The instructor will review the document and provide feedback.

Goal: Produce a clean, shareable record of the week’s learning.

⇒ **Homework blue** und **In-class practice exercises violet**

Before-Phase

Sie haben formal 28 Stunden fürs Selbststudium, aber davon ist vieles Klausurvorbereitung und Selbstfindung, so dass Sie zusätzlich jede zweite Woche 2 Stunden der Vorlesung erhalten

Meine Empfehlung: machen Sie die Stoffaneignung alleine und dann das Lösen der Hausaufgaben in festen Gruppen an diesen Tagen oder wann anders zusammen

Scheuen Sie sich nicht davor, eigene Ressourcen zu finden und dann der gesamten Gruppe zur Verfügung zu stellen.

Goals for today

You are able

to preprocess text: e.g., tokenize, normalize, lemmatize, remove stopwords, and compute n-grams/frequencies.

to implement this in spaCy: load models; use Doc/Token/Span; run POS/deps/NER; build Matcher rules.

Text Processing

From Bits to Knowledge…

Level	Example	Meaning
Bitstring	`012C0700` (hex) → `0000…0000` (bin)	Raw machine data
ASCII	`'1' ',' '7' '0'`	Characters
Syntax	`1,70`	Formatted number
Float (Data)	`1.70`	Numeric value
Semantics	Exchange rate	Interpreted concept
Context	EUR → USD	Real-world link
Information	"Rate is 1.70 EUR/USD"	Usable statement
Knowledge	"€100 → $170"	Applied decision

Preprocessing (Parsing)

Formats: Office docs, PDF, email, XML/HTML, images, audio, video, …

Goal: Make textual content processable → preprocessing/parsing

- Audio/Video → text (speech recognition)
- Image/Video → text (OCR)

Tools: Many commercial & open-source (e.g., Apache Tika)

Parser choice depends on:

- File format (txt, pdf, docx, xlsx, html, png, jpeg, mp4, …)
- Character set (UTF-8, CP1252, …)
- Language (English, …)

Detection methods: metadata, classification (algorithms/heuristics)

Overview Basic Text Processing Methods

Step	Action	Example
Regex Search	Rule-based pattern matching	Find dates, emails, etc.
Sentence Segmentation	Split text into sentences	`"This is one. That is two."` → 2 sentences
Tokenization	Split into words/tokens	`"cats running"` → `["cats","running"]`
Token Filtering	Remove noise: punctuation, numbers, stopwords	`"the, a, to"` → removed
Normalization	Standardize word forms	`"U.S.A." = "USA"`
Lemmatization / Stemming	Reduce to root form	`"authorize", "authorization"` → `"author"`

Regular expressions

Symbol	Meaning	Example
`.`	Any character	`a.b` → `acb`
`\d`	Digit `[0-9]`	`\d\d` → `42`
`\w`	Word char `[a-zA-Z0-9_]`	`\w+` → `hello`
`\s`	Whitespace	`\s` → space/tab
`*`	0 or more repetitions	`a*` → "", "a", "aa"
`+`	1 or more repetitions	`a+` → "a", "aa"
`?`	0 or 1 occurrence	`a?` → "", "a"
`[]`	Character set	`[abc]` → "a"

Sentence segmentation

Punctuation-based

- !, ?, : → usually unambiguous
- . → ambiguous (could be abbreviation, number, or EOS)

Ambiguity sources

- Abbreviations: Dr., Inc.
- Numbers: 4.3, .02%
- Initials

Solution: Binary classifier

- Decides EndOfSentence vs NotEndOfSentence
- Methods: regex, hand-written rules, machine learning

Punkt algorithm

- Builds dynamic dictionary of abbreviations, numbers, initials from text
- Every other . not in dictionary → EOS

Tokenization: Word-level

Break text into tokens (meaningful character sequences).

Input: "Friends, Romans and Countrymen." → Tokens: Friends , Romans and Countrymen.

Heuristic: split at whitespace + punctuation (aka "whitespace tokenizer").

Implementation: often regex-based, language-dependent, compiled into finite state machine.

Limitations:

- fixed rules → errors on new/rare words;
- large token vocabulary may hurt ML models.

Tokenization: Subwords

Goal: fixed-size vocabulary; frequent words = whole tokens, rare words = sub-word splits.

Components:

1. Token learner → builds vocabulary from training data.
2. Token segmenter → tokenizes text according to vocabulary.

Example: BERT → WordPiece tokenizer.

Byte Pair Encoding (BPE):

1. Start with characters as tokens (V0 = {a,b,…})
2. Merge most frequent adjacent tokens → new token
3. Replace sequences in training data, update vocabulary
4. Repeat until desired vocabulary size reached

Notes: usually run within words, not across word boundaries; first whole-word, then sub-word tokenization.

Token Filtering

Token Filtering

Remove tokens that reduce task performance

Examples: document metadata, headers/footers, URIs

Methods: blacklist, rules, heuristics

Stop Word Filtering

Stop words: little semantic content (the, a, and…)

High frequency (~30% of top 30 words)

Exclude to reduce noise and save space

Sometimes necessary to keep:

- Phrase queries: “King of Denmark”
- Titles: “Let it be”, “To be or not to be”
- Relational queries: “flights to London”

Lemmatization

Reduce words to base/dictionary form

- e.g., am, are, is → be
- car, cars, car's, cars' → car
- the boy's cars are different colors → the boy car be different color

Requires proper linguistic/morphological analysis

Stemming

Stemming: crude affix chopping to reduce terms to "roots"

- e.g., *automate(s), automatic, automation* → *automat, compressed*, *compression* → *compress*
- Simpler & faster than lemmatization
- IR evaluation: modest benefits of lemmatization vs. stemming

Most common English algorithm: Porter Stemmer

- Conventions + 5 sequential reduction phases
- Each phase: set of commands; choose rule matching longest suffix

Porter Rules examples:

- *sses* → *ss*
- *ies* → *i*
- *ational* → *ate*
- *tional* → *tion*

Word-sensitive rules: e.g., *replacement* → *replac* and *cement* → *cement*

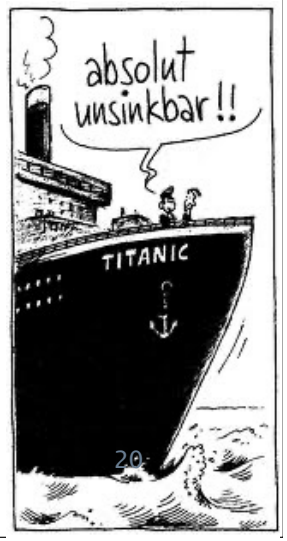
Linguistic Terminology

Term	Meaning
Syllables	phonological subunits of words
Morphemes	smallest meaningful parts of words (stem, affixes)
Word stem	the part of the word that is common to all its inflected variants
Word lemma	canonical/dictionary form of a word
Word form	the fully inflected word
Word class	verb, noun, adjective
Word types	distinct words in a text
Word tokens	each word in a text
N-grams	a sequence of n items from a text; items may be letters, syllables, words, …

Homework assignment

(All, 20 minutes)

Read the subsections 2.7.1-2.7.3 from https://web.stanford.edu/~jurafsky/slp3/2.pdf. It describes regular expressions. Explain how they can be used to find ICD-10 codes in a medical texts and what disadvantages in implies.



SpaCy

What is SpaCy?

- Industrial-strength NLP library in Python – designed for production use, not just research
- Provides pre-trained pipelines for tasks like tokenization, part-of-speech tagging, lemmatization, dependency parsing, and named entity recognition (NER).
- Supports custom model training and integration with deep learning frameworks (PyTorch, TensorFlow).
- Widely used in academia and industry for text processing, information extraction, and building NLP applications (chatbots, search engines, document analysis).

```
pip install spacy
python -m spacy download en_core_web_sm
```

Small model for demos; use *md/lg/trf* for higher accuracy

spaCy Processing Pipeline

Central object: Doc, with Token and Span

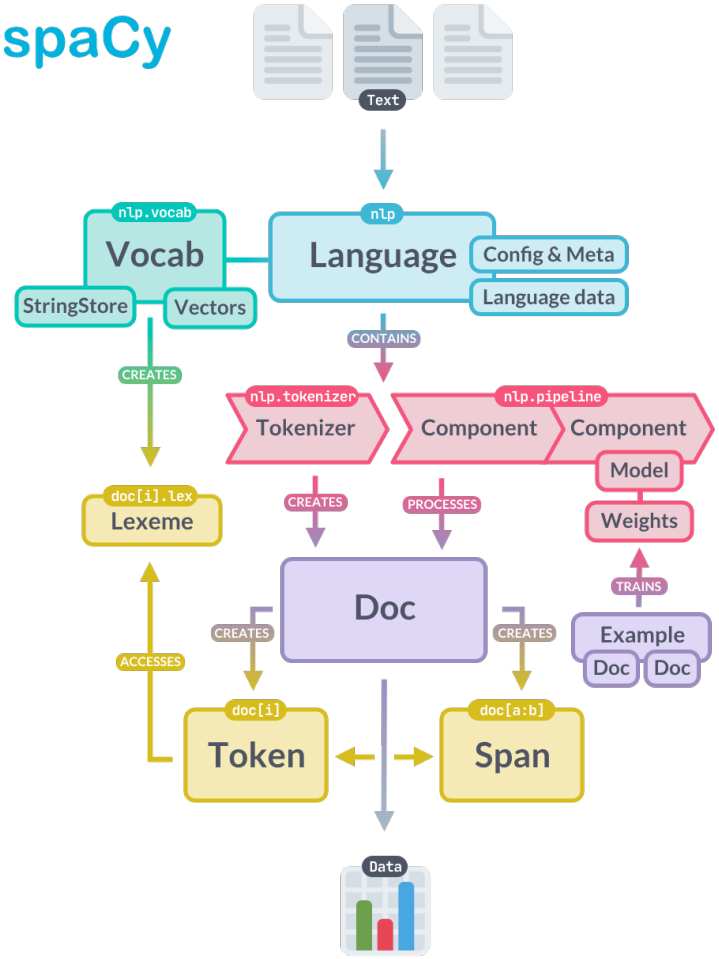
Input: raw text string

Output: Doc object with tokens, sentences, POS, deps, entities

Pipeline components run **in order**:

- Tokenizer → Tagger → Parser →
- NER → custom components

Each component updates the Doc



The Tokenizer

Splits text into tokens based on rules & exceptions
Handles punctuation, contractions, special cases
Produces Doc with Token objects

```
nlp = spacy.load("en_core_web_sm")
doc = nlp("Dr. Smith can't attend.")
[t.text for t in doc] # ["Dr.", "Smith", "ca", "n't", "attend", "."]
```

What does spacy.load("en_core_web_md") do?

- It loads a pre-trained English NLP pipeline (the medium-sized model, md).
- This pipeline comes with rules, statistical models, and word vectors.
- It prepares spaCy so you can run nlp(text) to process English text.
- It includes tokenization, tagging, parsing, lemmatization, NER, and semantic similarity via **Word vectors** (300-dimensional GloVe-like embeddings)

The Vocab

- Central storage of lexical information
- Contains:
  - String-to-hash mapping (Lexeme)
  - Word attributes: lowercase, shape, is_stop, etc.
- Shared across pipeline components for efficiency

The Doc Object

- Container for the entire text and annotations
- Key structures:
  - Token: one word/substring
  - Span: slice of tokens, e.g., a sentence or entity
- Acts as the core data structure passed through pipeline

Part-of-Speech Tagger

- Assigns POS tags (NOUN, VERB, etc.) to tokens
- Uses statistical model (pretrained on large corpora)
- Useful for syntax, lemmatization, and rule-based patterns

Dependency Parser

Identifies syntactic relations between words
Builds a tree: subject, object, modifiers

```
doc = nlp("The cat sat on the mat.")
[(t.text, t.dep_, t.head.text) for t in doc]
```

Named Entity Recognizer (NER)

Labels spans of text as entities: PERSON, ORG, DATE, etc.
Statistical + rule-based
Extendable with EntityRuler

```
doc = nlp("The cat sat on the mat.")
[(ent.text, ent.label_) for ent in doc.ents]
```

Hybrid strategy:

1. Regex for structured substrings (IDs, dates, emails)
2. Matcher/PhraseMatcher for lexical patterns
3. NER/Deps for linguistic relations

BIO Tagging for NER 1

What it is: A token-level scheme to mark entity Beginnings, Insides, or Outside any entity.

Tags:

- B-TYPE – first token of an entity span of class TYPE
- I-TYPE – subsequent token(s) of the same TYPE
- O – token not part of any entity

Core rules:

- Every entity starts with B-. Single-token entity ⇒ B-TYPE only.
- I-TYPE must follow a B-TYPE/I-TYPE of the same TYPE without gaps.
- Adjacent entities (even same TYPE) each restart with B-.
- No overlaps or nesting in a single BIO layer.
- Discontinuous mentions aren't representable; annotate contiguous spans only

BIO Tagging for NER 2

Mini examples:

“severe chest pain today”
→ B-SYM I-SYM I-SYM O

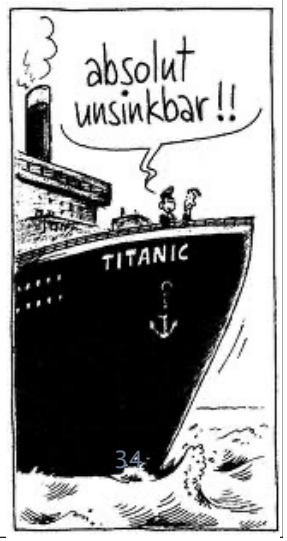
“pain in left arm”
→ O O B-ANAT I-ANAT

“arm and leg weakness” (two ANAT entities)
→ B-ANAT O B-ANAT O

Homework assignment

(All, 20 minutes)

Solve the three excercises in the Jupyter notebook Spacy-excercises.ipynb in the materials.



Text Processing with spaCy

Text processing components

- Search via **Regular Expressions** vs. **Rule-based** (Matcher/PhraseMatcher)
- **Sentence segmentation**
- **Tokenization**
- **Token filtering** (punctuation, numbers, dates, stop words)
- **Normalization** (case, Unicode; *U.S.A.* vs *USA*)
- **Lemmatization vs. Stemming** (*authorize* → *author*)

Search via Regular Expressions

Use Python re for format-constrained strings
Pros: simple, fast, no pipeline needed
Cons: no linguistic context; fragile with language variation

```
import re
text = "Contact: jane.doe@uni.edu, due: 2025-09-30"
emails = re.findall(r"[\w.+-]+@[\w.-]+\.[A-Za-z]{2,}", text)
dates = re.findall(r"\b\d{4}-\d{2}-\d{2}\b", text)
```

Rule-Based Search with spaCy Matcher

Matcher: token-level patterns using lexical & linguistic attrs
Robust to spacing/punctuation/token boundaries

```
from spacy.matcher import Matcher
nlp = spacy.load("en_core_web_sm")
matcher = Matcher(nlp.vocab)
pattern = [{"LOWER": {"IN": ["u.s.a.", "usa"]}}]
matcher.add("USA_VARIANTS", [pattern])
doc = nlp("He moved to the U.S.A. (USA) in 2020.")
for mid, start, end in matcher(doc):
span = doc[start:end]
print("match:", span.text)
```

PhraseMatcher for Fast Lookup

Ideal for **gazetteers** (product names, diseases, legal terms)

```
from spacy.matcher import PhraseMatcher
terms = ["machine learning", "natural language processing", "data mining"]
patterns = [nlp.make_doc(t) for t in terms]
pm = PhraseMatcher(nlp.vocab, attr="LOWER")
pm.add("TERMS", patterns)
doc = nlp("We teach Natural Language Processing.")
[doc[s:e].text for _, s, e in pm(doc)]
```

Tips:
Use attr="LOWER" or LEMMA for robustness
Build patterns once, reuse; serialize lists
Combine with SpanRuler to label custom entities

Sentence Segmentation

Task: cut character sequence into sentences
Challenges: abbreviations (Dr., U.S.A.), quotes, headlines, bullets
Two routes: (1) rule-based sentencizer, (2) statistical parser-based

```
# Rule-based: fast, punctuation-based
nlp = spacy.load("en_core_web_sm", exclude=["parser"])
if "senter" not in nlp.pipe_names:
nlp.add_pipe("sentencizer")
doc = nlp("Dr. Smith lives in the U.S.A. He teaches NLP.")
for sent in doc.sents:
print(sent.text)
```

Or keep parser for more accurate sentence boundaries

Tokenization

Task: cut character sequence into tokens (words)
Non-trivial: contractions (don't), hyphens, emojis, URLs, measure units
spaCy uses language-specific rules + exceptions

```
doc = nlp("U.S.A.-based co. raised $3.5M on 01/02/2025")
[(t.text, t.is_punct, t.like_num) for t in doc]
```

Inspect boundaries; adjust with Token attributes

Token Filtering (Punct., Numbers, Dates)

```
import re
from dateutil import parser as dparser # optional
keep = []
for t in doc:
if t.is_space or t.is_punct:
continue
if t.like_num:
continue
if re.fullmatch(r"\d{4}-\d{2}-\d{2}", t.text):
continue
keep.append(t)
```

Adapt filters to your task (don’t over-filter!)

Stop Word Filtering

```
from spacy.lang.en.stop_words import STOP_WORDS
[t for t in doc if t.text.lower() not in STOP_WORDS and not t.is_punct]
# Customize:
STOP_WORDS |= {"via", "based"}
```

Stopwords are **task-dependent**; revisit after modeling

Normalization

Map different word forms so equivalent forms align
Casefolding: "USA" == "usa"
Unicode normalization: accents/forms
Punctuation bridging: U.S.A. and USA should match

```
import unicodedata, re
raw = "U.S.A. – USA – u.s.a"
text = unicodedata.normalize("NFKC", raw)
text = text.lower()
text = re.sub(r"\bu\.s\.a\.\b", "usa", text)
text = re.sub(r"[^\w\s]", " ", text)
" ".join(text.split())
```

Keep a **normalization map** for known variants

Lemmatization vs. Stemming

Lemmatization: vocab-aware (better grammar): authorizes → authorize
Stemming: heuristic truncation (authorize, authorization → author)
Choose based on evaluation, not dogma

```
doc = nlp("They were authorizing the authorizations.")
[(t.text, t.lemma_, t.pos_) for t in doc]
# e.g., ("authorizing", "authorize"), ("authorizations", "authorization")
```

Stemming (if needed)

spaCy **does not** include a stemmer; use NLTK if stemming required

```
from nltk.stem import PorterStemmer
import nltk
nltk.download('wordnet')

# Stemming
stemmer = PorterStemmer()
tokens_stemmed = [stemmer.stem(t) for t in tokens_filtered]
tf_stemmed = Counter(tokens_stemmed)
print(tf_stemmed)
```

Homework assignment

(All, 10 minutes)

Discuss briefly why and when to use a stemmer, when one can have lemmatizer?