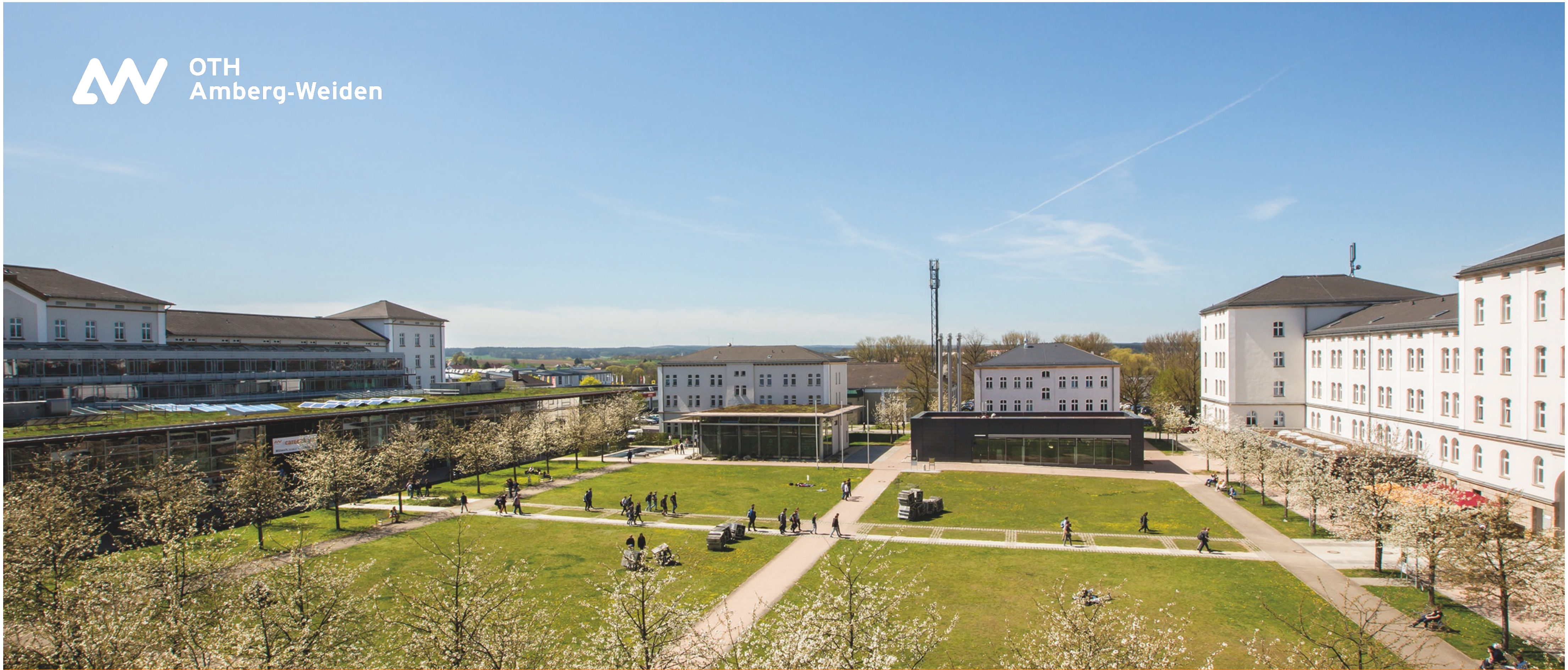


NATURAL LANGUAGE PROCESSING

WORDS AND SENTENCES

What is a Word?

Easy Question?

What are words?

OTH Amberg-Weiden

- “This is my cat, I call her Kitty.”
  - This sentence has 8 words.
  - How do you treat punctuation (here: ”,” and “.”)? If you count it as words the sentence will have 10 words.
- Now consider speech applications. You have a transcript of spoken language:
  - “I um do large- largely data uh processing work.”
  - How do you treat filled pauses like “um” or “uh” sounds? Are these words?
  - How do you treat fragments like “large-”? Is it counted as a separate word?
- For speech applications there may be good reasons to keep fragments or filled pauses or to filter them out.

What are words?

- “This is my cat, I call her Kitty. Do you have cats, too?”
  - Here we have the words “cat” and “cats”
  - Both have the same word stem “cat”, also called lemma.
  - The two words have the same lemma but different wordform
  - Examples
    - Nouns in singular and plural have the same lemma but typically different wordforms
    - Verbs depending on time or grammatical form: show, showing, showed

How Would You Identify a Sentence?

Practical Part

- Go back to your notebook NLP01 and identify all sentences in The Time Machine.
- Do not use sentence tokenizers from any packages, yet.
- Implement your ideas.
- Keep a record of your sentences.
- How many sentences did you identify?

Organization of Texts

OTH Amberg-Weiden

- In NLP a piece of text is usually called a document
- It contains words organized in sentences.
  - A sentence is usually a series of words finished by a characteristic punctuation: fullstop (“.”), question marke (“?”), exclamation mark (“!”)
    - Difficult to treat the colon (“:”)
    - “Then she said: ‘I never come back.’ → Two sentences
    - “He wrote his list: milk, honey, eggs, bread.” → one sentence.
  - However, exceptions exist:
    - Not every fullstops marks the end of a sentence, e.g. “e.g.” or “H.G. Wells”)
    - Some sentences do not end with “.” but e.g. with certain kinds of whitespaces. Typical example: Headlines

Organization of Texts

- Splitting according to punctuation needs to consider exceptions:
  - Not every fullstops marks the end of a sentence, e.g. “e.g.” or “H.G. Wells”)
  - IP addresses 10.0.0.1
  - Prices $39.99
  - Some sentences do not end with “.” but e.g. with certain kinds of whitespaces. Typical example: Headlines

Text Processing

- Before we can analyze any text we usually need the following preprocessing steps:
  - Sentence tokenization (splitting the text into sentences)
  - Word tokenization (splitting the text into words)
  - Word normalization (reducing the words to their corresponding lemmata)

Text Processing

OTH Amberg-Weiden

- Before we can analyze any text we usually need the following preprocessing steps:
  - **Sentence tokenization** (splitting the text into sentences)
  - Word tokenization (splitting the text into words)
  - Word normalization (reducing the words to their corresponding lemmata)

Sentence Tokenization

- We have explored regular expressions before.
- Regular expression could be used to subdivide a document into sentences.
- However, you have to account for all the exceptions mentioned before…
- … and those we haven’t considered yet…

- Good sentence tokenization requires more sophisticated methods.

Unsupervised Sentence Tokenization Algorithm

- T. Kiss and J. Strunk, Unsupervised Multilingual Sentence Boundary Detection. Computational Linguistics 32: 485-525 (2006)
- Paper: https://aclanthology.org/J06-4003.pdf
- The algorithm is based on classification of occurrences of a fullstop as
  - Abbreviation (street "St.", abbreviation "abbrev.", for example "e.g.", …)
  - Ellipsis (…)
  - Initials (H.G. Wells)
  - Ordinal numbers (1. Chapter, I. Chapter)
- Special case: Abbreviation at the end of a sentence ("I live at Porter St.")

Sentence Boundary Detection
Idea

- The algorithm classifies the occurrences of fullstops in several steps
  - First, classify into abbreviation, ellipsis, sentence end.
- In the second stage these finds are refined:
  - Abbreviations are checked whether they coincede with a sentence end, being reclassified as sentence end.
  - Ellipses are checked whether they coincede with a sentence end, being reclassified as sentence end.
  - Checking whether a sentence end is acutally an initial, being reclassified as abbreviation.
  - Checking whether a sentence end is acutally an ordinal number, being reclassified as abbreviation.

Sentence Boundary Detection
Likelihood Ratios

- The algorithm is based on likelihood ratios
  - How likely is a point to appear behind a certain word.
  - Normal words appear in any part of a sentence and usually not only at the sentence end.
  - Therefore, the likelihood ratio of word followed by fullstop / word not followed by fullstop is small.
  - Abbreviations are certain to be followed by a fullstop.

Sentence Boundary Detection
Heuristics

- In the second classification stage three heuristics are used
  - orthographic heuristic
  - collocation heuristic
  - frequent sentence starter heuristic

Sentence Boundary Detection
Heuristics

- Orthographic heuristic
  - After a sentence end, the following word is written capitalized.
  - Problems:
    - Some words are usually written capitalized independent of a sentence ending, e.g. names, “I”, …
    - Some words are never written capitalized (e.g. mathematical equations)
  - Use likelihood ratios: Count whether the word in question is written at least once in lower case.
  - Example: “I don’t like abbrev. because they make trouble. Peter likes abbrev. They make his life easier.”
    - In this sample the first occurrence of abbrev. is in the middle of a sentence. The second one at the end of a sentence.
    - They is written capitalized in the second case indicating a sentence end.
    - This is valid since “they” occurs at least once uncapitalized, therefore is not always written capitalized.

Sentence Boundary Detection
Heuristics

- Collocation heuristic
  - Abbreviations often show collocations left and right of the fullstop
  - E.g. “e.g.”, “i.e.”, “et. al.”
  - Sentence boundary typically do not show collocation.
  - If there is a high likelihood ratio for the words left and right of the fullstop it is an abbreviation rather than a sentence boundary.

Sentence Boundary Detection
Heuristics

- Frequent sentence starter heuristic
  - Determine words which frequently occur at the beginning of a sentence.
  - Use part of the text where the sentence boundary is sure.

- For benchmarks and further details, please read the original paper.

Text Processing

- Before we can analyze any text we usually need the following preprocessing steps:
  - Sentence tokenization (splitting the text into sentences)
  - **Word tokenization** (splitting the text into words)
  - Word normalization (reducing the words to their corresponding lemmata)

Word Tokenization

- Basically, word tokenization means splitting text into single words.
- Can be done pattern based using regular expressions.
- Typical pattern for a word [A-Za-z]+
- Typical patterns indicating the end of a word:
  - Punctuation
  - End of string
  - Whitespaces

Word tokenization

- Typical problems in word tokenization:
  - A period is not necessarily the end of a sentence
    - IP addresses 127.0.0.1
    - Dates (in Europe) 10.10.2023
    - Decimal numbers (in English) $ 9.99
    - Ellipsis
  - Clitic contractions
    - don’t, doesn’t = do not
    - I’ll = I will
    - …

NLTK

- The Python-based NLP toolkit NLTK offers several word tokenizers.
- nltk.regexp_tokenize()

```
>>> text = 'That U.S.A. poster-print costs $12.40...'
>>> pattern = r'''(?x)     # set flag to allow verbose regexps
...     (?:[A-Z]\.)+       # abbreviations, e.g. U.S.A.
...   | \w+(?:-\w+)*       # words with optional internal hyphens
...   | \$?\d+(?:\.\d+)?%? # currency and percentages, e.g. $12.40, 82%
...   | \.\.\.             # ellipsis
...   | [][.,;"'?():-_`]   # these are separate tokens; includes ], [
... '''
>>> nltk.regexp_tokenize(text, pattern)
['That', 'U.S.A.', 'poster-print', 'costs', '$12.40', '...']
```

Code from Bird et. al., NLTK Book, Chapter 3

Text Processing

- Before we can analyze any text we usually need the following preprocessing steps:
  - Sentence tokenization (splitting the text into sentences)
  - Word tokenization (splitting the text into words)
  - **Word normalization** (reducing the words to their corresponding lemmata)

Word Normalization

- **Normalization** → Putting into standard format representing multipe forms
  - **Lemmatization** → Putting each word to its root, for example
    - am, are, is → be (irregular verbs)
    - Word, words → word (plural forms)
  - **Case folding** → putting everything in a single way of use of capital letters, for example
    - The Time Machine → the time machine (all to lower case)

Lemmatization
Algorithms

- Most lemmatization is based on decomposing words into parts (**morphemes**)
  - The main part of the word is the so-called **stem**.
  - Other parts are called **affixes**.
  - E.g. plural dogs = morpheme “dog” (stem) + morpheme “s” (affix)
  - E.g. plural affixes = “affix” (stem) + “es” (affix)
- Normalization usually means reducing each word to its stem.
  - Therefore, lemmatization often is referred to as **stemming**.

Lemmatization
Algorithms

- A problem with stemming, it is not particularly well defined.
- Many irregular cases
  - women → woman
  - lying → lie
- A rather old, but still used, algorithm for lemmatization is the **Porter stemmer.**
- Alternative: **Lancaster stemmer** (other rule set)
- Alternative: **WordNet lemmatizer**
  - Removes affixes only if the resulting word is in its vocabulary.
  - Slower performance due to this comparison.

Porter Stemmer

- Porter stemmer uses stemming rules it applies to words.
  - The rules are applied in sequence (5 steps).
- Examples for stemming rules:
  - Step 1: Plural and Participals
    - SSES -> SS | IES -> I | SS -> SS | S ->
    - caresses -> caress | ponies -> poni | caress -> caress | cats -> cat
  - Step 2
    - ATIONAL -> ATE relational -> relate | TIONAL -> TION conditional -> condition, rational -> ration
- Find more here: https://tartarus.org/martin/PorterStemmer

https://tartarus.org/martin/PorterStemmer/def.txt

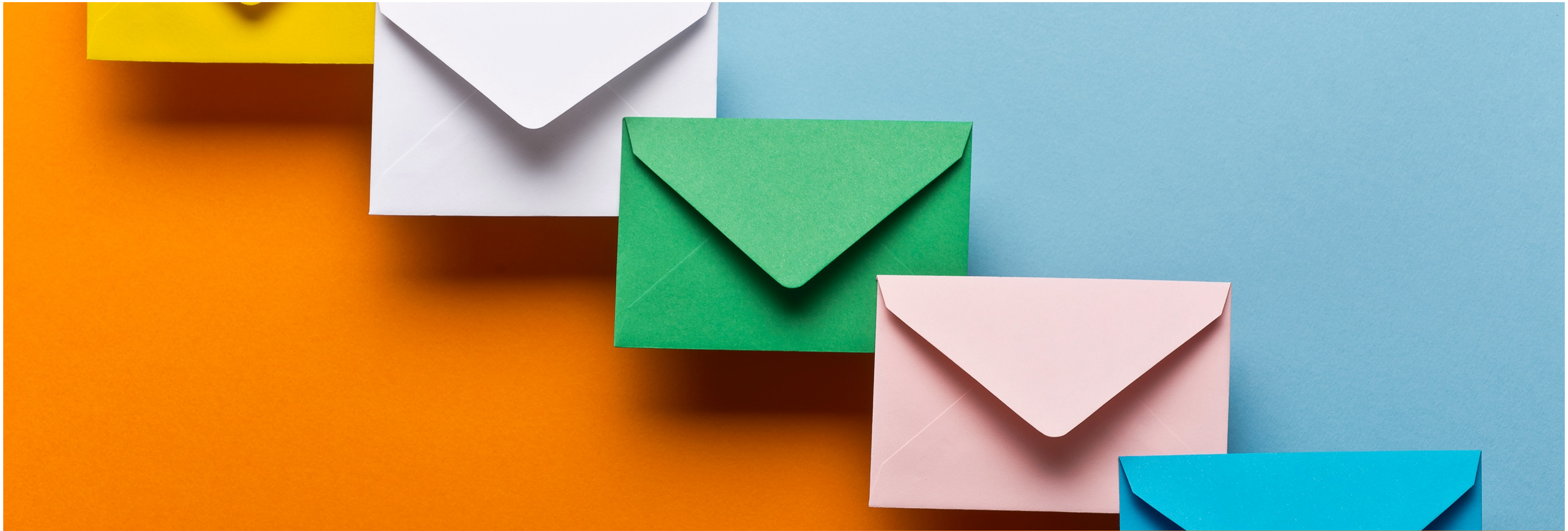
Stemming – General

OTH Amberg-Weiden

- Choose a stemmer that fits your application.
  - Do you search in the text and want to find alternative forms of search words?
    - In this case Porter stemmer is a valid choice.
  - Do you require valid words as lemmas?
    - In this case the WordNet lemmatizer is preferrable.

Practical Part

- Please follow the section about stemming in Notebook NLP02



DISTANCES BETWEEN STRINGS

String Comparison

- **Comparing two strings** is a usual operation.
- Consider **“inention”**, which is obviously not a complete word but contains some error.
  – Maybe, the author wanted to write “intention”
  – A mere string comparison will fail ”inention” == “intention” → False
  – Maybe the author meant to write “intervention”? Or “invention”? Or “mention”?
- What is the difference between the strings “inention”, “intention”, ”intervention”, “invention”, ”mention”, …
- A more complicated example: “OTH president Prof. Bulitta” vs. “OTH president Bulitta”
  – Both strings mean the same person, however, they are not equal.

Edit Distance

- Edit Distance is a distance measure based on the number of elemental string operations to transform one string into another one.
- Elemental string operations:
  - Insertion
  - Deletion
  - Substitution
- Alignment: Correspondence between two substrings.
  - Needs to be considered for this kind of string comparison.

Alignment
Example

- Different alignments for our example “inention” and “invention”:

I	N	V	E	N	T	I	O	N
I	N	E	N	T	I	O	N	*
*	I	N	E	N	T	I	O	N
I	N	*	E	N	T	I	O	N

Edit Distance

- Levenshtein distance: Count the number of insertion and deletion steps to transform one string into another.
- No substitution operation, however, substitution = 1 deletion + 1 insertion

- Looking at the alignment: We can compute various distances between “inention” and “invention”

Alignment
Levenshtein distance

- Different alignments lead to different Levenshtein distances for our example “inention” and “invention”:

I	N	V	E	N	T	I	O	N	
I	N	E	N	T	I	O	N	*	13
*	I	N	E	N	T	I	O	N	5
I	N	*	E	N	T	I	O	N	1

Edit Distance

- To determine the distance between two strings, we are looking for their **minimum edit distance**.
- Compare two strings of (not necessarily equal) lengths n and m.
  - Brute force search will not work for longer strings due to complexity
  - n > m: $\binom{n}{m}$ can get become a very large number, much larger than n.
- Efficient algorithm works with recurrence relation
  - Let D(i,j) be the distance between string1[0:i] and string2[0:j]
  - $D(i,j) = min\begin{Bmatrix} D(i-1,j) + delete\_cost(string1[i]) \\ D(i,j-1) + insert\_cost(string2[j]) \\ D(i-1,j-1) + substitution\_cost(string1[i], string2[j]) \end{Bmatrix}$

Minimum Edit Distance Algorithm

```
function MIN-EDIT-DISTANCE(source, target) returns min-distance

  n ← LENGTH(source)
  m ← LENGTH(target)
  Create a distance matrix D[n+1,m+1]

  # Initialization: the zeroth row and column is the distance from the empty string
  D[0,0] = 0
  for each row i from 1 to n do
     D[i,0] ← D[i-1,0] + del-cost(source[i])
  for each column j from 1 to m do
     D[0,j] ← D[0,j-1] + ins-cost(target[j])

  # Recurrence relation:
  for each row i from 1 to n do
        for each column j from 1 to m do
           D[i,j] ← MIN( D[i−1,j] + del-cost(source[i]),
                         D[i−1,j−1] + sub-cost(source[i], target[j]),
                         D[i,j−1] + ins-cost(target[j]))
  # Termination
  return D[n,m]
```

From the book of Jurafsky and Martin (Figure 2.17)

Literature
Zweite Headline

- Daniel Jurafsky, James H. Martin , Speech and Language Processing, Copyright © 2023. All rights reserved. Draft of January 7, 2023.
  Online: https://web.stanford.edu/~jurafsky/slp3/ed3book_jan72023.pdf
- Steven Bird, Ewan Klein, and Edward Loper, Natural Language Processing with Python – Analyzing Text with the Natural Language Toolkit, O-Reilly, 3rd edition (2019), Online Version https://www.nltk.org/book/
- https://docs.python.org/3/howto/regex.html#regex-howto

Quellen: