UNIVERSITÄT
ZU KÖLN

COUNTING WORDS, CORPUS STATISTICS, ENCODING

Sprachverarbeitung (Vorlesung)

Janis Pagel*

17 April 2025

*Based on slides by Nils Reiter

Recap

- Computational Linguistics as a discipline between computer science and linguistics
  - also known as "natural language processing", (NLP)
  - Experiments are important way of making progress in CL
- Corpora

01

CORPORA

Corpora

- (Large) collections of linguistic expressions
- Speech corpora: Spoken language
  - File formats: wav, mp3, ...
- Text corpora: Written language
  - File formats: txt, xml, json, ...

Corpora

- (Large) collections of linguistic expressions
- Speech corpora: Spoken language
  - File formats: wav, mp3, ...
- Text corpora: Written language
  - File formats: txt, xml, json, ...
- Why do we look at corpora?

Corpora

- (Large) collections of linguistic expressions
- Speech corpora: Spoken language
  - File formats: wav, mp3, ...
- Text corpora: Written language
  - File formats: txt, xml, json, ...
- Why do we look at corpora?
  - Making statements about language needs to take into account many language expressions
  - We under-estimate creativity, flexibility and productivity of language use

  $\rightarrow$ Empiricism

Meta data and annotations

Meta data: Data about the data

- Information about the corpus
- Language, date of creation, author(s), publication source, ...
- Machine-readable: XML, JSON, CSV, ...

Meta data and annotations

Meta data: Data about the data

- Information about the corpus
- Language, date of creation, author(s), publication source, ...
- Machine-readable: XML, JSON, CSV, ...

Annotations: Data about parts of the corpus

- Examples
  - Linguistic annotation: Parts of speech, named entities, syntactic relations, ...
  - Non-linguistic annotation: Sentiment expressions, rhetoric devices, arguments, ...

Meta data and annotations

Meta data: Data about the data

- Information about the corpus
- Language, date of creation, author(s), publication source, ...
- Machine-readable: XML, JSON, CSV, ...

Annotations: Data about parts of the corpus

- Examples
  - Linguistic annotation: Parts of speech, named entities, syntactic relations, ...
  - Non-linguistic annotation: Sentiment expressions, rhetoric devices, arguments, ...
- Explicit location in the corpus: Document/word/character numbers in text, milliseconds in speech

Preparations (for text corpora)

- OCR: Optical Character Recognition (MS99, p. 123)
  - Convert images (e.g., from a scan) into text
  - Huge improvements in last five years

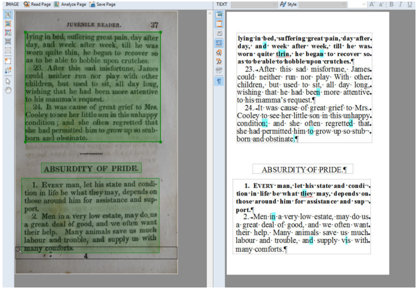

Figure: Source

Preparations (for text corpora)

- OCR: Optical Character Recognition (MS99, p. 123)
  - Convert images (e.g., from a scan) into text
  - Huge improvements in last five years

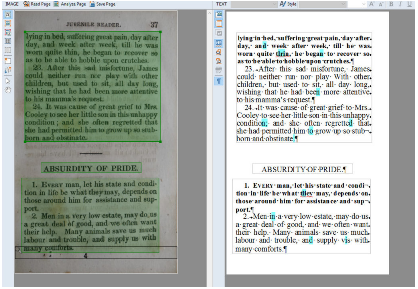

Figure: Source

- Encoding: How to specify characters in a computer
  - Simple: ASCII (7 bit per character, $2^7 = 128$ different characters)
  - Outdated: Latin-1 / ISO-8859 (8 bit, $\Rightarrow 256$ diff. characters)
  - Modern: Unicode (e.g., UTF-8)
    - $1\,\mathrm{B/char}$ to $4\,\mathrm{B/char}$
    - $1\,112\,064$ characters can be represented

Tools and Techniques

- Plain text editors
  - We often want to inspect the corpus as it is on disk (i.e., without an editor interfering too much)
  - Mac: Textmate/emacs/vi; Windows: Notepad++/emacs/vi

Tools and Techniques

- Plain text editors
  - We often want to inspect the corpus as it is on disk (i.e., without an editor interfering too much)
  - Mac: Textmate/emacs/vi; Windows: Notepad++/emacs/vi
- Regular expressions
  - The most important tool for corpus analysis
    - Cleanup (e.g., after scraping a corpus from the web)
    - Analysis (e.g., to find all variants of a word or deal with slang)
  - Usable in *all** programming languages and find tools

Tools and Techniques

- Plain text editors
  - We often want to inspect the corpus as it is on disk (i.e., without an editor interfering too much)
  - Mac: Textmate/emacs/vi; Windows: Notepad++/emacs/vi
- Regular expressions
  - The most important tool for corpus analysis
    - Cleanup (e.g., after scraping a corpus from the web)
    - Analysis (e.g., to find all variants of a word or deal with slang)
  - Usable in *all** programming languages and find tools
- Command line
  - Large corpora often cannot be displayed with GUI tools
  - Command line tools faster and more memory efficient

Tokenization

- Segmenting a corpus into individual units
- Tokens: Words, punctuation, numbers, symbols, ...

Tokenization

- Segmenting a corpus into individual units
- Tokens: Words, punctuation, numbers, symbols, ...
- Naive: Splitting at white space (space, newline, ...)
  - Why naive?

Tokenization

- Segmenting a corpus into individual units
- Tokens: Words, punctuation, numbers, symbols, ...
- Naive: Splitting at white space (space, newline, ...)
  - Why naive?
- Solved, but complex
  - E.g., syntactic points vs. morphological points
- Sometimes, shortcuts are ok – depends on the use case

Word Counts

Count	Word
585	die
584	und
407	er
404	der
348	zu
311	sich
259	nicht
250	sie
243	in
243	den
233	war
218	Gregor
189	mit
178	das
176	auf
171	es
162	dem
155	hatte
137	ein
136	aber
133	daß
123	als
110	auch
107	Schwester
	...

Word Counts

Count	Word
585	die
584	und
407	er
404	der
348	zu
311	sich
259	nicht
250	sie
243	in
243	den
233	war
218	Gregor
189	mit
178	das
176	auf
171	es
162	dem
155	hatte
137	ein
136	aber
133	daß
123	als
110	auch
107	Schwester
	...

- Number of words in a text
- Most frequent words (MFW) are function words
- ‘Content words’ that appear often indicate text content

Word Counts

Count	Word
585	die
584	und
407	er
404	der
348	zu
311	sich
259	nicht
250	sie
243	in
243	den
233	war
218	Gregor
189	mit
178	das
176	auf
171	es
162	dem
155	hatte
137	ein
136	aber
133	daß
123	als
110	auch
107	Schwester
	...

- Number of words in a text
- Most frequent words (MFW) are function words
- ‘Content words’ that appear often indicate text content

Stop Word Removal

- Common practice: Remove “stop words”
- But there are choices:
  - Should stop words be removed at all?
  - Which words do we consider stop words?
- Removing words is not content-preserving!

Zipf's Law

MS99, pp. 23 sqq.

- George Kingsley Zipf (1902-1950): American Linguist
- Basic property of human language
  - Frequency distribution of words (in a corpus) is stable
  - Word frequency is inversely proportional to its position in the ranking

$$f \propto \frac{1}{r}$$

(there is a constant k, such that $f \times r = k$)

Zipf's Law

MS99, pp. 23 sqq.

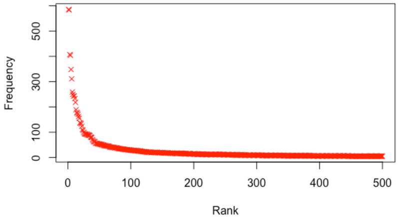

Figure: Words sorted after their frequency (red). Text: Kafka's "Die Verwandlung".

Zipf's Law

MS99, pp. 23 sqq.

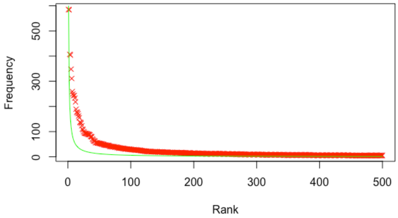

Figure: Words sorted after their frequency (red). Zipf distribution: $y = 600\frac{1}{x}$ (green). Text: Kafka's "Die Verwandlung".

Zipf's Law

MS99, pp. 23 sqq.

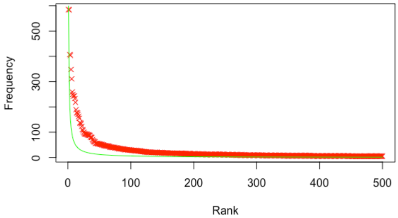

Figure: Words sorted after their frequency (red). Zipf distribution: $y = 600\frac{1}{x}$ (green). Text: Kafka's "Die Verwandlung".

Consequences

- Very few words appear with very high frequency
- The vast majority of words appear only once (Hapax Legomena)
  - It's difficult to learn something about these words!

Counting Words

- Absolute numbers are not that interesting
- Insights are only generated through comparison

Abs. number	Word form
20	women
67	woman
31	men
79	family
82	sister
83	friend
99	bath
117	father
133	man
144	sir

Table: Jane Austen's *Persuasion* (nouns)

Abs. number	Word form
0	friend
2	bath
11	women
23	men
30	father
68	woman
83	family
113	sir
121	man
282	sister

Table: Jane Austen's *Sense and Sensibility* (nouns)

Absolute Numbers

Word	Persuasion	Sense
woman	67	68
women	20	11
man	133	121
men	31	23
sister	82	282

...does it make sense to compare absolute numbers? No.

Absolute Numbers

Word	Persuasion	Sense
woman	67	68
women	20	11
man	133	121
men	31	23
sister	82	282

...does it make sense to compare absolute numbers? No.

- The texts/corpora do not have the same size
- Scaling using their length: Division by the total number of words

Absolute Numbers

Word	Persuasion		Sense	
woman	67	$0.000\,79\,\%$	68	$0.000\,55\,\%$
women	20	$0.000\,24\,\%$	11	$0.000\,09\,\%$
man	133	$0.001\,58\,\%$	121	$0.001\,00\,\%$
men	31	$0.000\,37\,\%$	23	$0.000\,19\,\%$
sister	82	$0.000\,97\,\%$	282	$0.002\,33\,\%$

...does it make sense to compare absolute numbers? No.

- The texts/corpora do not have the same size
- Scaling using their length: Division by the total number of words
- Visible changes: Proportion of “sister”: $3.4 \rightarrow 2.4$

Scaling

- Number of words: Result of a measurement
- If measuring in different scenarios, it's important to scale the results
  - "In a text that is much shorter, there are much less chances for a certain word to be used."

Scaling

- Number of words: Result of a measurement
- If measuring in different scenarios, it's important to scale the results
  - "In a text that is much shorter, there are much less chances for a certain word to be used."

Recipe

- Divide the result of the measurement by the theoretical maximum
- How many chances are there for "sister" to be used?
  - As many as there are words in the text
- Thus, we divide by the total number of words

Scaling

- Number of words: Result of a measurement
- If measuring in different scenarios, it's important to scale the results
  - "In a text that is much shorter, there are much less chances for a certain word to be used."

Recipe

- Divide the result of the measurement by the theoretical maximum
- How many chances are there for "sister" to be used?
  - As many as there are words in the text
- Thus, we divide by the total number of words

- It's not always obvious how to scale
- When reading research: Was it scaled, and how?

Types and Tokens

Manning and Schütze (MS99, pp. 21 sq.)

- If a text has been tokenized, we can access individual units: Tokens
- Not all tokens are words: Punctuation, detached prefixes, ...

Types and Tokens

Manning and Schütze (MS99, pp. 21 sq.)

- If a text has been tokenized, we can access individual units: Tokens
- Not all tokens are words: Punctuation, detached prefixes, ...
- We are often also interested in different tokens: Types

Types and Tokens

Manning and Schütze (MS99, pp. 21 sq.)

- If a text has been tokenized, we can access individual units: Tokens
- Not all tokens are words: Punctuation, detached prefixes, ...
- We are often also interested in different tokens: Types

Example

the cat chases the mouse

Types and Tokens

Manning and Schütze (MS99, pp. 21 sq.)

- If a text has been tokenized, we can access individual units: Tokens
- Not all tokens are words: Punctuation, detached prefixes, ...
- We are often also interested in different tokens: Types

Example

the cat chases the mouse

- Tokens: the, cat, chases, the, mouse
- Types: the, cat, chases, mouse

Type-Token-Ratio (TTR)

- What is the relation between number of tokens and number of types?

Type-Token-Ratio (TTR)

- What is the relation between number of tokens and number of types?
- Construct a sentence with 5 tokens and 5 types!

Type-Token-Ratio (TTR)

- What is the relation between number of tokens and number of types?
- Construct a sentence with 5 tokens and 5 types!
  - "the dog barks loudly ."

Type-Token-Ratio (TTR)

- What is the relation between number of tokens and number of types?
- Construct a sentence with 5 tokens and 5 types!
  - "the dog barks loudly ."
- Construct a sentence with 5 tokens and 4 types!

Type-Token-Ratio (TTR)

- What is the relation between number of tokens and number of types?
- Construct a sentence with 5 tokens and 5 types!
  - “the dog barks loudly .”
- Construct a sentence with 5 tokens and 4 types!
  - “the cat loves the mouse”

Type-Token-Ratio (TTR)

- What is the relation between number of tokens and number of types?
- Construct a sentence with 5 tokens and 5 types!
  - “the dog barks loudly .”
- Construct a sentence with 5 tokens and 4 types!
  - “the cat loves the mouse”
- Construct a sentence with 5 tokens and 1 type!

Type-Token-Ratio (TTR)

- What is the relation between number of tokens and number of types?
- Construct a sentence with 5 tokens and 5 types!
  - "the dog barks loudly ."
- Construct a sentence with 5 tokens and 4 types!
  - "the cat loves the mouse"
- Construct a sentence with 5 tokens and 1 type!
  - "dog dog dog dog dog" (not really a sentence ...)
  - It's not possible to create a 'proper' sentence with 1 type

Type-Token-Ratio (TTR)

- Measure for 'lexical variability'

$$TTR = \frac{\text{number of types}}{\text{number of tokens}}$$

- Max value: 1

Type-Token-Ratio (TTR)

- Measure for 'lexical variability'

$$TTR = \frac{\text{number of types}}{\text{number of tokens}}$$

- Max value: 1 (there cannot be more types than tokens)
- Min value: $\epsilon = \frac{1}{\text{very large number}}$

Type-Token-Ratio (TTR)

- Measure for 'lexical variability'
$$TTR = \frac{\text{number of types}}{\text{number of tokens}}$$
- Max value: 1 (there cannot be more types than tokens)
- Min value: $\epsilon = \frac{1}{\text{very large number}}$
- Real (German) texts
  - $10\,000$ words (Wikipedia): $\frac{4021}{10\,000} = 0.4021$

TTR and Text Length

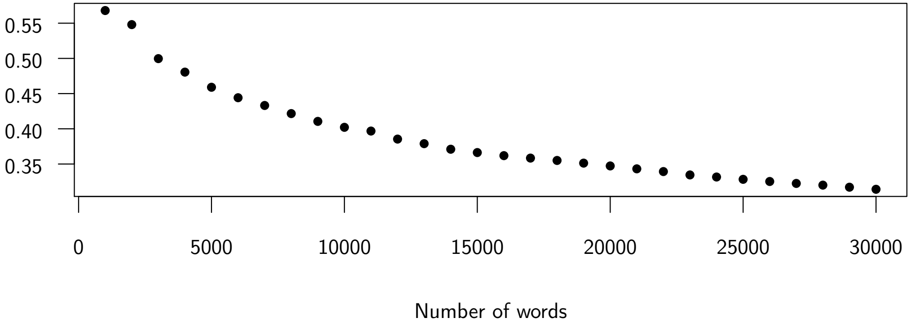

Figure: Type-Token-Ratio for increasing text lengths

TTR and Text Length

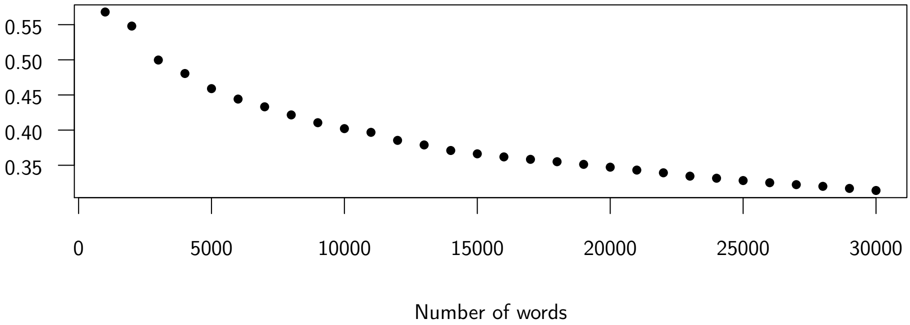

Figure: Type-Token-Ratio for increasing text lengths

- Increasing length $\rightarrow$ lower TTR!
- Why?

TTR and Text Length

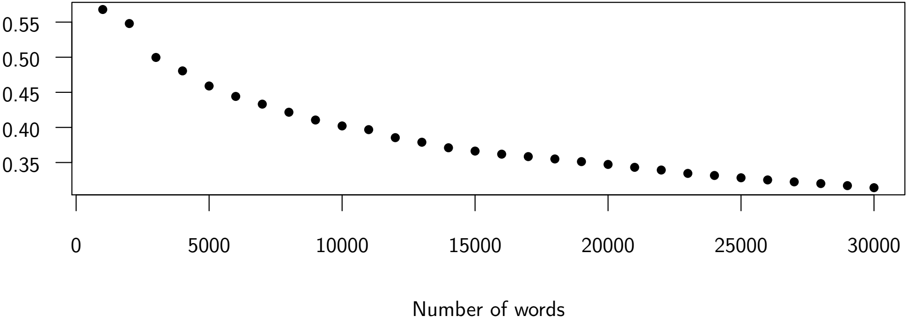

Figure: Type-Token-Ratio for increasing text lengths

- Increasing length → lower TTR!
- Why?– Zipf!

Standardized TTR (STTR)

- Calculate TTR over windows of fixed size (e.g., 1000 words)
- Calculate arithmetic mean over TTR values

Standardized TTR (STTR)

- Calculate TTR over windows of fixed size (e.g., 1000 words)
- Calculate arithmetic mean over TTR values

$$TTR_n = \frac{\text{number of types in } n\text{th window}}{\text{number of tokens in } n\text{th window}}$$

Standardized TTR (STTR)

- Calculate TTR over windows of fixed size (e.g., 1000 words)
- Calculate arithmetic mean over TTR values

$$
\begin{aligned}
TTR_n &= \frac{\text{number of types in } n\text{th window}}{\text{number of tokens in } n\text{th window}} \\
STTR &= \frac{1}{w}\sum_{i=0}^{w} TTR_i
\end{aligned}
$$

- n is the window size
- w is the number of windows
- i is the current index of the window calculated
- $\sum$ is the symbol for a sum

n-grams

- So far: Individual tokens
- But: Context is important for linguistic expressions

n-grams

- So far: Individual tokens
- But: Context is important for linguistic expressions
- n-gram: A list of n directly adjacent tokens
  - Popular choices for n: 2 to 4

n-grams

- So far: Individual tokens
- But: Context is important for linguistic expressions
- n-gram: A list of n directly adjacent tokens
  - Popular choices for n: 2 to 4

Example

The dog barks.

- 1-grams: "the", "dog", "barks", "."
- 2-grams (bigrams): "the dog", "dog barks", "barks ."
- 3-grams (trigrams): "the dog barks", "dog barks ."

02

ENCODING

Introduction

- How to represent text data in a computer
- Enumeration: Each character is assigned a number
- American Standard Code for Information Interchange (ASCII)
  - $128 = 2^7$ characters, including control symbols for telegraphy
  - No German Umlauts etc.

Wikipedia: ASCII

Introduction

- How to represent text data in a computer
- Enumeration: Each character is assigned a number
- American Standard Code for Information Interchange (ASCII)
  - $128 = 2^7$ characters, including control symbols for telegraphy
  - No German Umlauts etc.
- Unicode: A single standard to represent *all* characters from all languages
  - $155\,063$ characters, including CJK ideographs
  - Complex enumeration scheme

Wikipedia: ASCII

unicode.org

Unicode 16.0 charts

Unicode

- Code point: An integer in the Unicode standard
  - Written in hexadecimal and prefixed with U+
  - E.g.: `U+00E4` = "Latin Small Letter a with Diaeresis" = ä

Unicode

- Code point: An integer in the Unicode standard
  - Written in hexadecimal and prefixed with U+
  - E.g.: `U+00E4` = “Latin Small Letter a with Diaeresis” = ä
- Mapping methods used to map each code point onto a code unit
  - Code unit: A sequence of bytes that represent some character
- Unicode transformation format (UTF): Most common mapping
  - UTF-8: uses one to four bytes for each code point, maximizes compatibility with ASCII
    - Default in Python
  - UTF-16, uses one or two 16-bit code units per code point
    - Strings in Java
  - UTF-32, uses one 32-bit code unit per code point

Unicode

UTF-8

- Code points U+0000 to U+007F (128) represented in ASCII way, with a leading zero
  - E.g.: $A_{\text{ASCII}} =$ U+0041 $= 65_{10} = 41_{16} = 1000001_2 =$ | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |

Unicode

UTF-8

- Code points U+0000 to U+007F (128) represented in ASCII way, with a leading zero
  - E.g.: $A_{\text{ASCII}} =$ U+0041 $= 65_{10} = 41_{16} = 1000001_2 =$ $\boxed{0}\boxed{1}\boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{1}$
- Code points U+0080 to U+07FF (1920) are represented in two bytes
  - First byte starts with 110, second with 10
  - E.g.: ä = U+00E4 $= 228_{10} = 11100100_2 =$ $\boxed{1}\boxed{1}\boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{1}\boxed{1}$ $\boxed{1}\boxed{0}\boxed{1}\boxed{0}\boxed{0}\boxed{1}\boxed{0}\boxed{0}$

Unicode

UTF-8

- Code points U+0000 to U+007F (128) represented in ASCII way, with a leading zero
  - E.g.: A_{ASCII} =U+0041 $= 65_{10} = 41_{16} = 1000001_2 =$ | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
- Code points U+0080 to U+07FF (1920) are represented in two bytes
  - First byte starts with 110, second with 10
  - E.g.: ä = U+00E4 $= 228_{10} = 11100100_2 =$ | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
- U+0800 to U+FFFF: | 1 | 1 | 1 | 0 | | | | | | 1 | 0 | | | | | | | | 1 | 0 | | | | | | | (three bytes)
- U+10000 to U+10FFFF: 4 Bytes, first one starting with 11110, others with 10

Unicode

Parsing UTF-8

- If a byte starts with a 0: The character is one byte long
- If a byte starts with a 1:
  - The number of 1s before the first 0 determine how many bytes belong to this character
    - This means the first byte starts with at least two 1s
  - Check that the following start with 10
  - Take them together as a single character

Unicode

Parsing UTF-8

- If a byte starts with a 0: The character is one byte long
- If a byte starts with a 1:
  - The number of 1s before the first 0 determine how many bytes belong to this character
    - This means the first byte starts with at least two 1s
  - Check that the following start with 10
  - Take them together as a single character
- Everything else is an undefined byte sequence
  - Maybe it's a different encoding?

Unicode

Parsing UTF-8

- If a byte starts with a 0: The character is one byte long
- If a byte starts with a 1:
  - The number of 1s before the first 0 determine how many bytes belong to this character
    - This means the first byte starts with at least two 1s
  - Check that the following start with 10
  - Take them together as a single character
- Everything else is an undefined byte sequence
  - Maybe it's a different encoding?

Determining Encoding

- It is difficult to (automatically) determine the encoding of a text
- "11000011 10100100" is "ä" in UTF-8, but "Ã¤" in ISO Latin 1 – how to know what's correct?

Unicode

Combined Characters

- For flexibility, there is a mechanism for combining characters
- `U+0300` to `U+036F` defines combining diacritical marks
- To be combined with the preceding character
- `U+0041 U+0308` represent "Ä" in decomposed form

Unicode

Combined Characters

- For flexibility, there is a mechanism for combining characters
- `U+0300` to `U+036F` defines combining diacritical marks
- To be combined with the preceding character
- `U+0041 U+0308` represent “Ä” in decomposed form
- `U+00C4` *also* represents “Ä” (in composed form)

Unicode

Combined Characters

- For flexibility, there is a mechanism for combining characters
- `U+0300` to `U+036F` defines combining diacritical marks
- To be combined with the preceding character
- `U+0041 U+0308` represent "Ä" in decomposed form
- `U+00C4` *also* represents "Ä" (in composed form)

Normalization

- Normalization Form D (NFD):
  - "Canonical Decomposition"
  - All combined characters are represented in their decomposed form
- Normalization Form C (NFC):
  - "Canonical Decomposition, followed by Canonical Composition"

Unicode

More (Interesting) Oddities

- Ω
  - Represented as `U+2126` and `U+03A9`

Unicode

More (Interesting) Oddities

- Ω
  - Represented as `U+2126` and `U+03A9`
  - `U+03A9`: The Greek letter
  - `U+2126`: The physical unit for electrical resistance

Unicode

More (Interesting) Oddities

- Ω
  - Represented as `U+2126` and `U+03A9`
  - `U+03A9`: The Greek letter
  - `U+2126`: The physical unit for electrical resistance
- Country Flags
  - Emoji support came 2010, including country flags
  - No individual code point for each flag
  - Instead: Regional indicator symbols that represent ISO 3166-1 codes for countries
  - Implementations should render `U+1F1E9 U+1F1EA` as 🇩🇪
    - If that's not possible, use Roman letters (`U+1F1E9 U+1F1EA` [🇩 🇪] = DE)

Unicode

More (Interesting) Oddities

- Ω
  - Represented as U+2126 and U+03A9
  - U+03A9: The Greek letter
  - U+2126: The physical unit for electrical resistance
- Country Flags
  - Emoji support came 2010, including country flags
  - No individual code point for each flag
  - Instead: Regional indicator symbols that represent ISO 3166-1 codes for countries
  - Implementations should render U+1F1E9 U+1F1EA as 🇩🇪
    - If that's not possible, use Roman letters (U+1F1E9 U+1F1EA [🇩 🇪] = DE)
- Emoji skin color variation: Similar to character combination
  - U+1F44C U+1F3FB = 👌🏻　　U+1F44C U+1F3FF = 👌🏿
  - U+1F44C: 👌 U+1F3FB: 🏻　　U+1F44C: 👌 U+1F3FF: 🏿

Unicode

More (Interesting) Oddities

- Ω
  - Represented as U+2126 and U+03A9
  - U+03A9: The Greek letter
  - U+2126: The physical unit for electrical resistance
- Country Flags
  - Emoji support came 2010, including country flags
  - No individual code point for each flag
  - Instead: Regional indicator symbols that represent ISO 3166-1 codes for countries
  - Implementations should render U+1F1E9 U+1F1EA as 🇩🇪
    - If that's not possible, use Roman letters (U+1F1E9 U+1F1EA [🇩 🇪] = DE)
- Emoji skin color variation: Similar to character combination
  - U+1F44C U+1F3FB = 👌🏻   U+1F44C U+1F3FF = 👌🏿
  - U+1F44C: 👌 U+1F3FB: 🏻   U+1F44C: 👌 U+1F3FF: 🏿
- "a" also represented twice
  - U+0061: Latin small letter a
  - U+0430: Cyrillic small letter a

⚠ This is/was also a security risk, because info@mybank.com and info@mybank.com look similar

Fun with Unicode

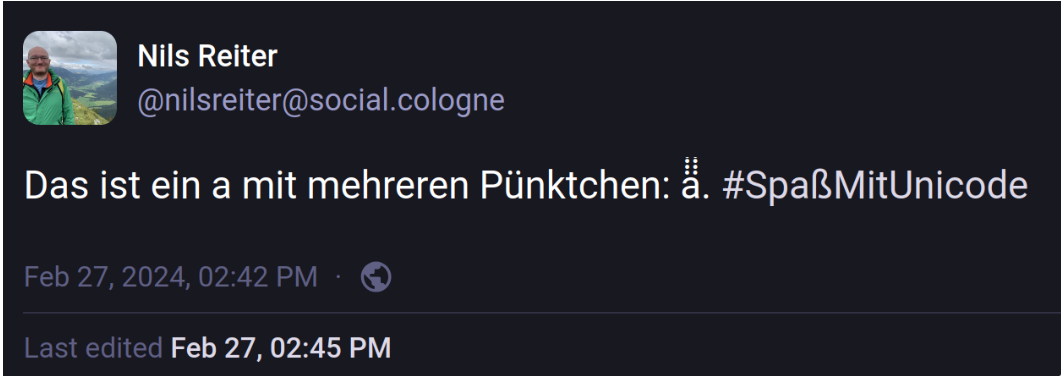

Figure: Source

03

SUMMARY

Summary

- Types and tokens
- Zipf distribution
- Type-Token-Ratio
- Encoding
- Unicode

References

Manning, Christopher D. and Hinrich Schütze (1999). *Foundations of Statistical Natural Language Processing*. Cambridge, Massachusetts and London, England: MIT Press.