344.075 KV: Natural Language Processing

Principles of Text Processing

Navid Rekab-Saz

navid.rekabsaz@jku.at

JOHANNES KEPLER
UNIVERSITY LINZ

Institute of
Computational
Perception

Agenda

- NLP terminology and low-level tasks
- Sparsity in language
- Text pre-processing

Agenda

- **NLP terminology and low-level tasks**
- Sparsity in language
- Text pre-processing

Natural Language

- Learned from experience
- A symbolic/discrete system …

 kangaroo

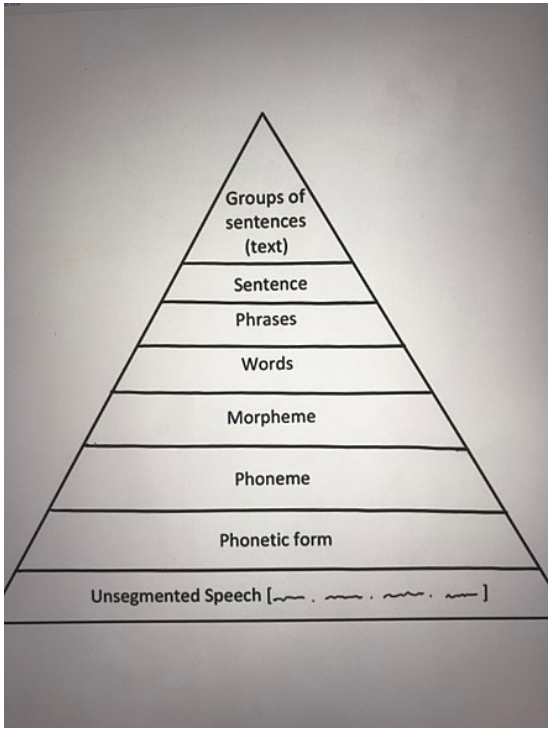
 volcano

- Encodings in brains and models are continuous

Language hierarchy – linguistic perspective

Sample sentence: "*a fox jumped over the lazy dog.*"

- Phonemes
  - A unit of sound, e.g. /p/, /t/ and /æ/
- Morphemes
  - Smallest meaningful unit in a word, e.g. the word ***national*** has two morphemes: ***nation*** a noun, and ***-al*** a suffix
- Words
- Phrase
  - Noun phrase: ***a fox***
  - Verb phrase: "***jumped over the lazy dog***"
- Sentence



https://en.wikipedia.org/wiki/Syntactic_hierarchy

Language hierarchy – computational perspective

Sample sentence: "*a fox jumped over the lazy dog.*"

- Character
  - like "***a***", "***f***", "***x***", etc.
- N-gram character
  - E.g. tri-gram characters like "***a f***", " ***fo***", "***fox***", and "***ox*** "
- Word
- N-gram word
  - E.g. tri-grams like "***a fox jumped***", "***fox jumped over***" , and "***jumped over the***"
- Compound noun
  - is made up of at least two nouns like ***post office***, ***San Francisco***
- Multiword Expression
  - Made up of at least two words like
    - ***hang up the gloves*** (idiom)
    - ***in short***

Language hierarchy – computational perspective (cont.)

Sample sentence: "*a fox jumped over the lazy dog.*"

- Token
  - used as the unit of processing
  - can be any of the previously mentioned units and any sequence of characters
  - E.g. when tokenized by words:
    - ["***a***", "***fox***", " ***jumped***", "***over***", "***the***", "***lazy***", "***dog***"]
- Dictionary or vocabulary list or lexicon
  - List of unique tokens in the given text
- Sentence
- Paragraph
- Document
- Corpus
  - a collection of text documents

Low-level tasks: Language Modeling

Look more:
https://talktotransformer.com
https://www.youtube.com/watch?v=gcHkxP9adiM

Low-level tasks: Natural Language Inference

Text	Judgments	Hypothesis
A man inspects the uniform of a figure in some East Asian country.	contradiction C C C C C	The man is sleeping
An older and younger man smiling.	neutral N N E N N	Two men are smiling and laughing at the cats playing on the floor.
A black race car starts up in front of a crowd of people.	contradiction C C C C C	A man is driving down a lonely road.
A soccer game with multiple males playing.	entailment E E E E E	Some men are playing a sport.
A smiling costumed woman is holding an umbrella.	neutral N N E C N	A happy woman in a fairy costume holds an umbrella.

https://nlp.stanford.edu/projects/snli/

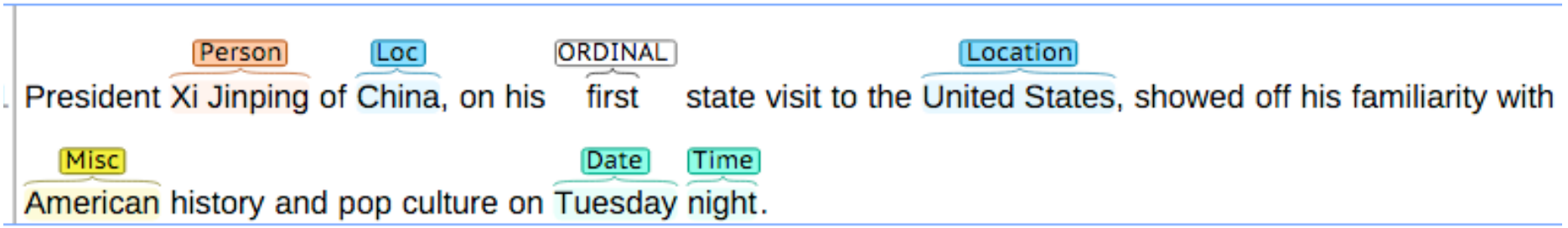
Low-level tasks: Word Sense Disambiguation

- 2 senses of ***bank***
  - "The ***bank*** will not be accepting cash on Saturdays. "
  - "The river overflowed the ***bank.***"

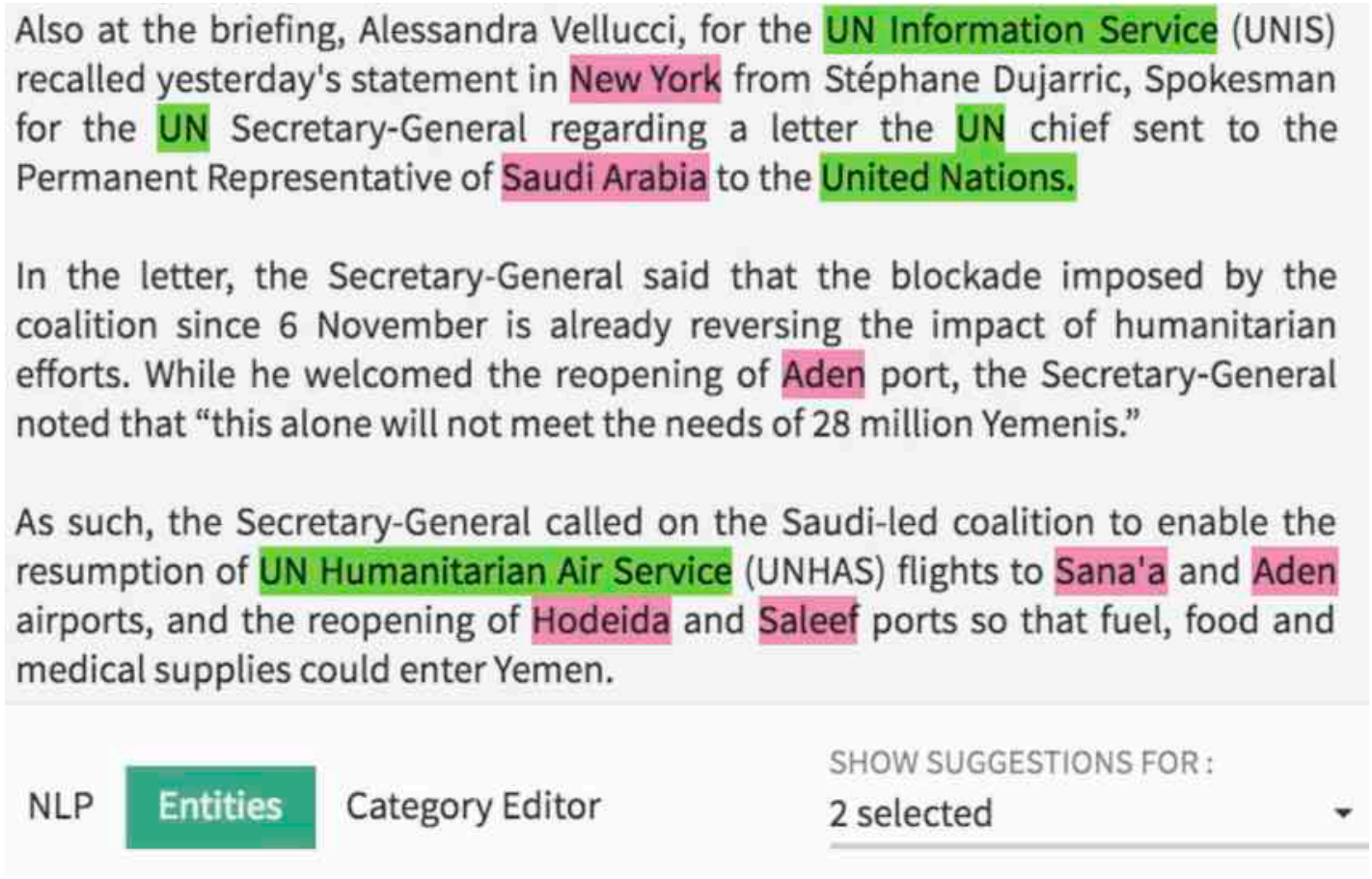
- 8 senses of ***bass***, as defined in WordNet
  - bass - (the lowest part of the musical range)
  - bass, bass part - (the lowest part in polyphonic music)
  - bass, basso - (an adult male singer with the lowest voice)
  - sea bass, bass - (flesh saltwater fish of the family Serranidae)
  - freshwater bass, bass - (any of various North American lean-fleshed freshwater fishes especially of the genus Micropterus)
  - bass, bass voice, basso - (the lowest adult male singing voice)
  - bass - (member with the lowest range of a family of musical instruments)
  - bass - (nontechnical name for any of numerous edible marine and freshwater spiny-finned fishes)

Low-level tasks: Named Entity Recognition (NER)

- “Named Entity Recognition labels sequences of words in a text which are the names of things, such as person and company names, or gene and protein names.”



Low-level tasks: Named Entity Recognition (NER)

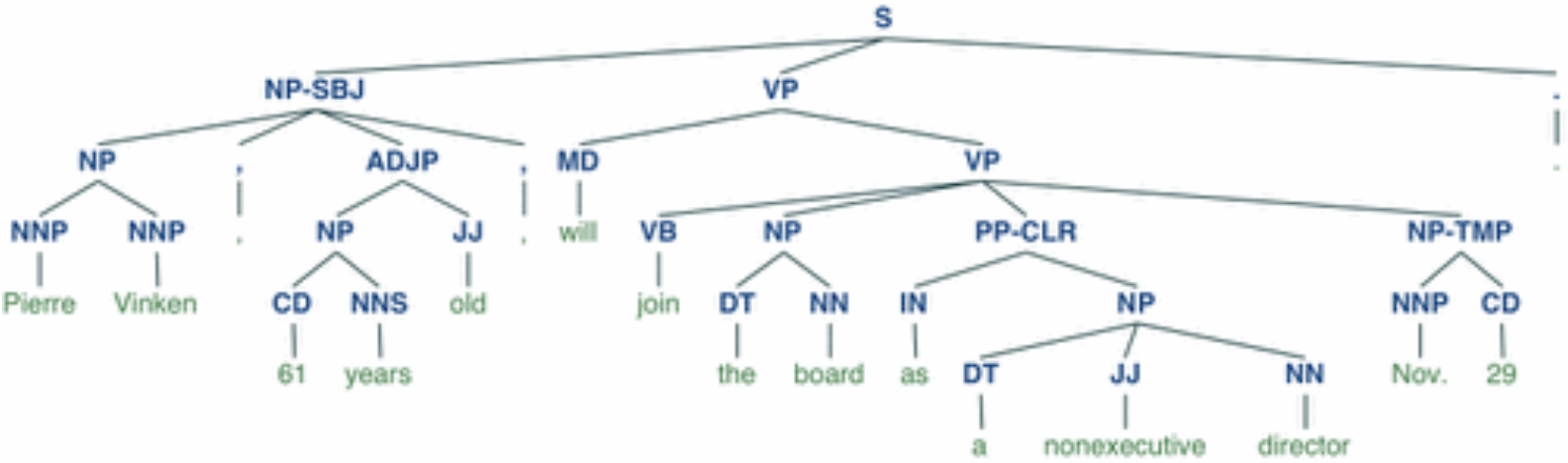


Locations

Organizations

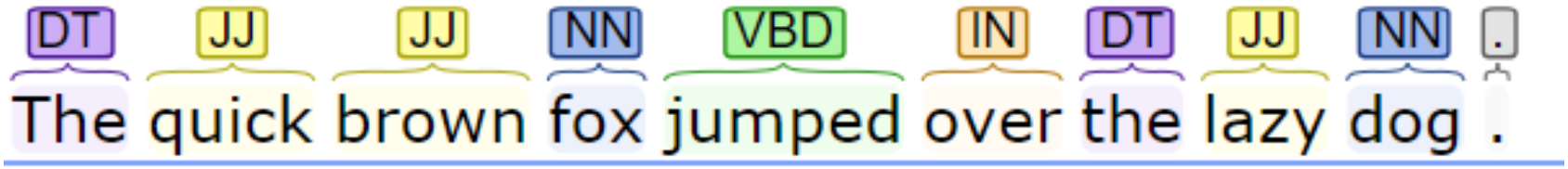
Low-level tasks: Parsing

- A natural language parser is a program that works out the grammatical **structure of sentences**, for instance, which groups of words go together (as "phrases") and which words are the **subject** or **object** of a verb



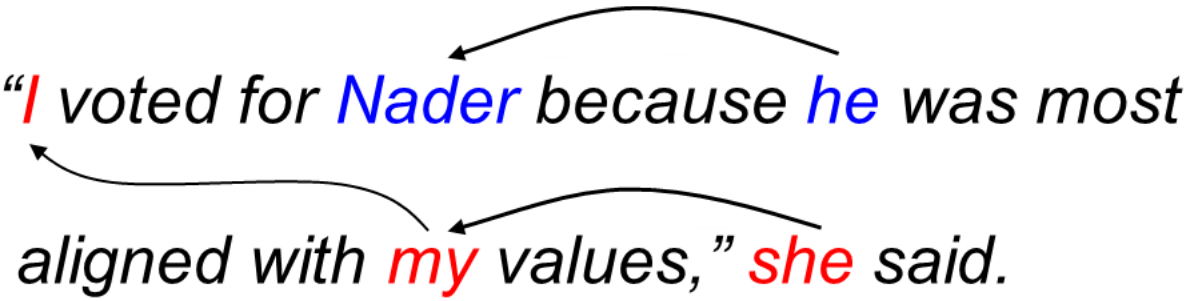
Low-level tasks: Part-of-Speech (POS) Tagging

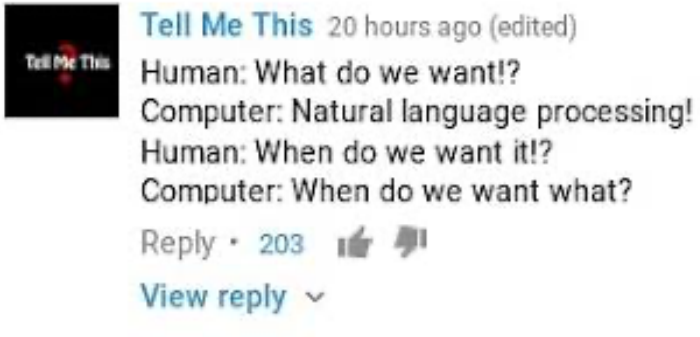
- “A Part-Of-Speech Tagger is a piece of software that reads text in some language and assigns parts of speech to each word, such as noun, verb, adjective”



Low-level tasks: Coreference Resolution

- Coreference resolution is the task of finding all expressions that refer to the same entity in a text





Agenda

- NLP terminology and low-level tasks
- **Sparsity in language**
- Text pre-processing

Challenges in language processing

- Language is a complex social process
- Tremendous ambiguity at every level of representation
- Variability
- **Sparsity**
- Grounding

Zipf's Law

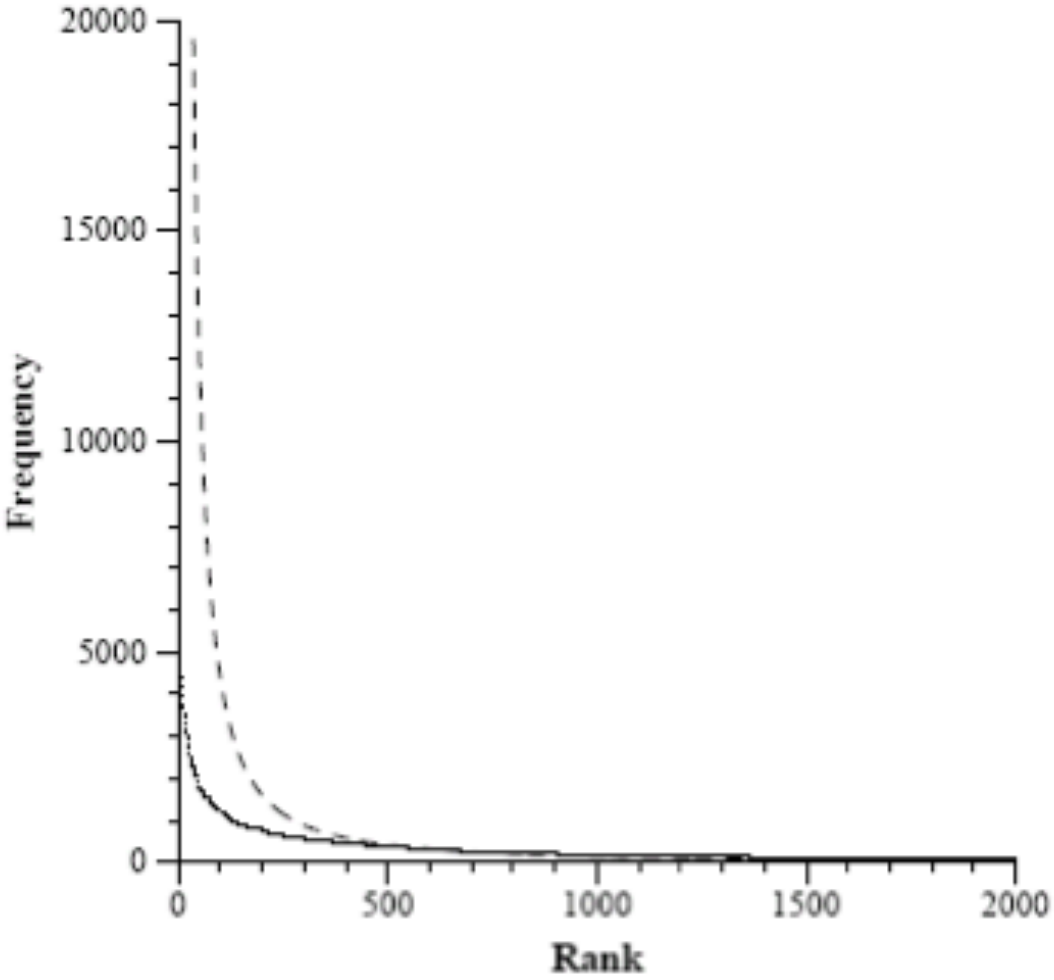
- Sort the vocabularies of a dictionary based on their frequencies in a text corpus
  - E.g. the results from the Brown Corpus:

Rank	Token	Frequency
1	the	22038615
2	be	12545825
3	and	10741073
4	of	10343885
5	A	10144200
…	…	…

- Top words in the list are called *stop words*
- The ones at the bottom are *rare words*

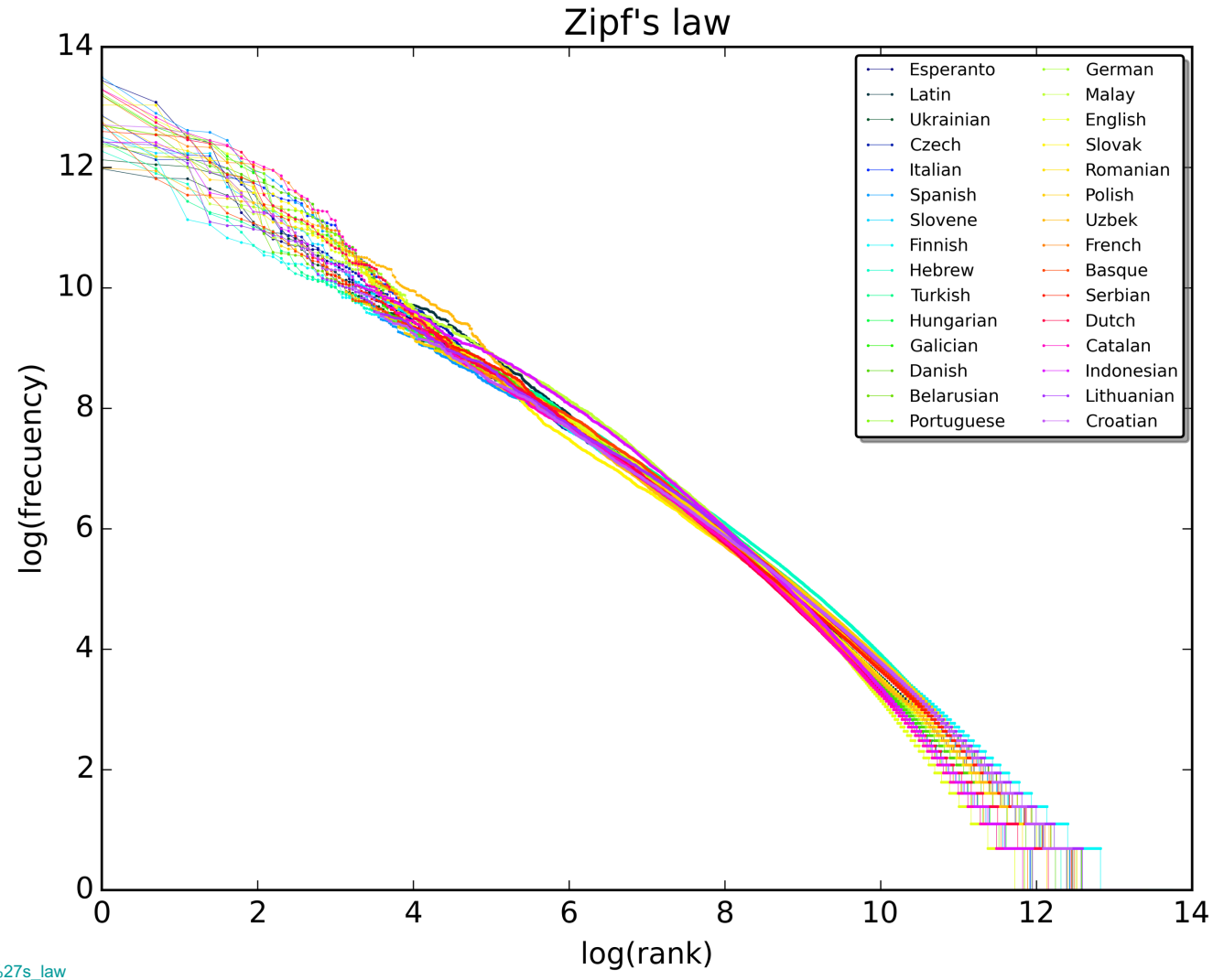
Zipf's Law

- The plot of rank versus frequency looks like this:



Zipf's Law

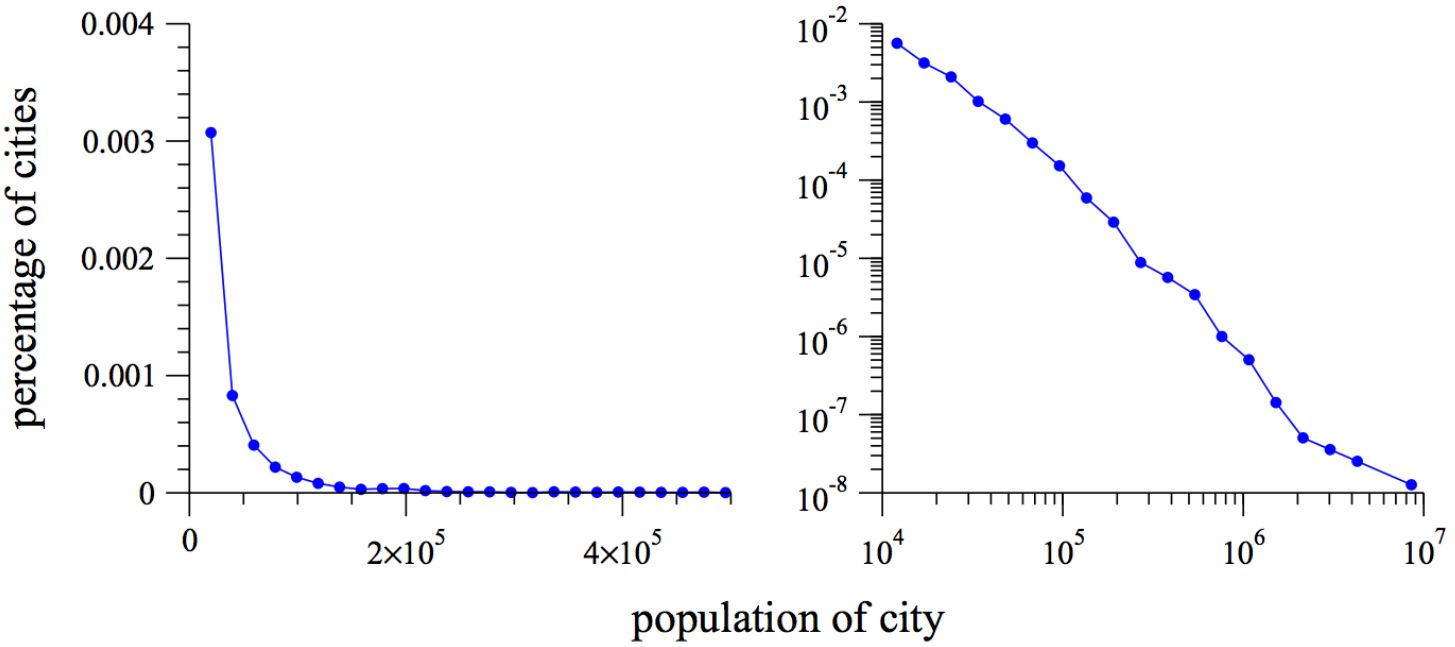
- Applying logarithm to both axes:



https://en.wikipedia.org/wiki/Zipf%27s_law

Power Law in social science

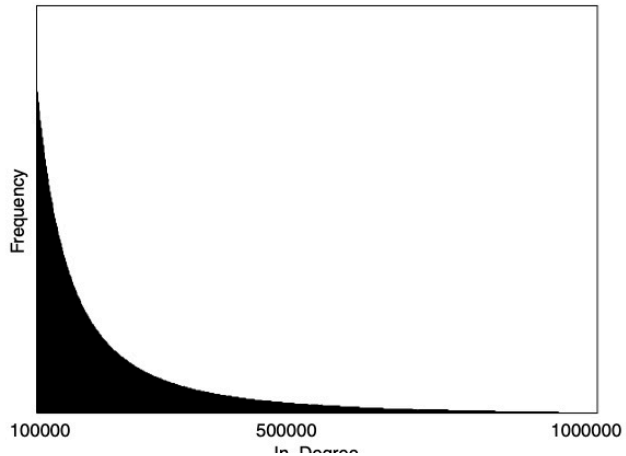
- Observed in other human-related phenomena/systems
  - Population rank of cities
  - Pagerank scores of websites
  - Income distribution
  - Corporation sizes

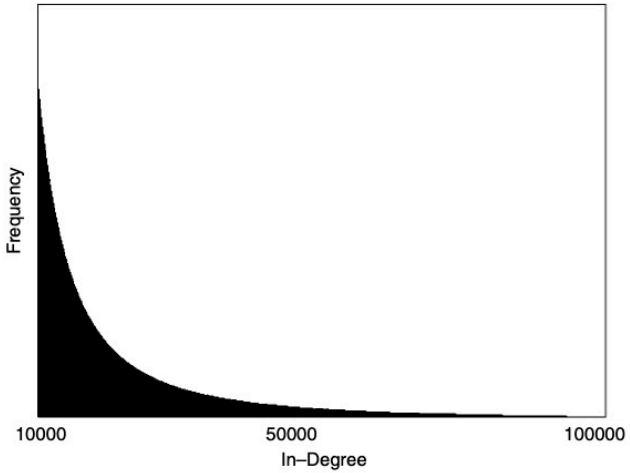


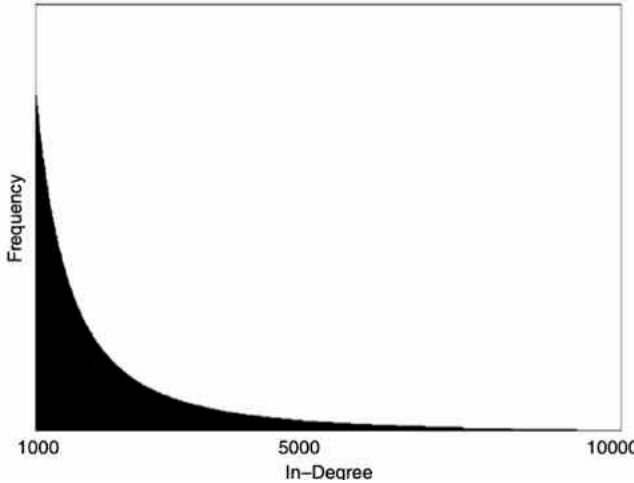
https://statweb.stanford.edu/~owen/courses/306a/ZipfByHera.pdf

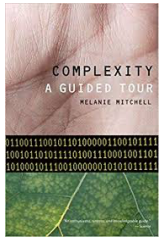
Power Law in web

- Approximation of the shape of web based on the number of incoming links to each website





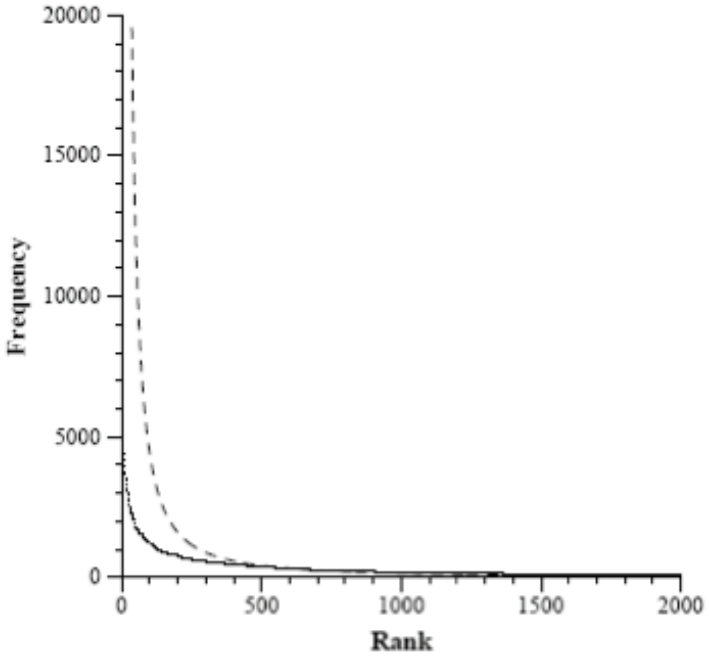




What does Zipf' Law tell us?

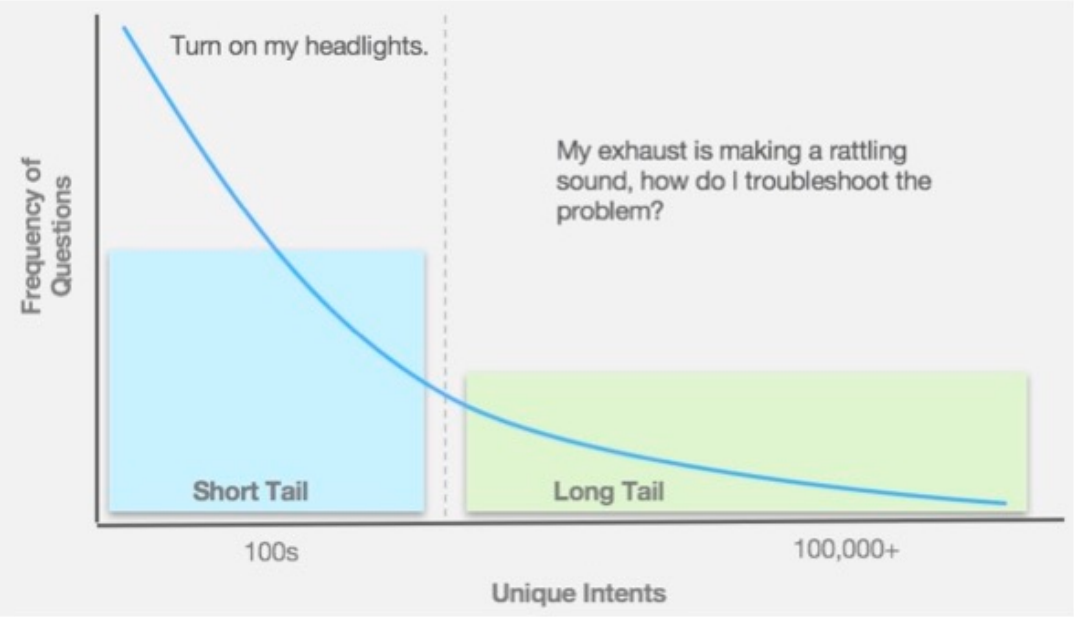
- Highly frequent tokens cover a large portion of a corpus
  - Only 135 most frequent words cover half of the Brown Corpus

- A large portion of the dictionary consists of words with very low frequencies
  - Typically removing words with frequencies of lower than 3 halves the size of dictionary!



What does Zipf' Law tell us?

- The challenge of long tail
  - Phenomena with low frequency (resided on the tail of distribution) are the challenging parts of language processing
  - Example of the long tail problem in a question answering system:



- Why could it be challenging for statistical models?

https://www.syntravlaanderen.be/sites/default/files/atoms/files/Conversational%20Bots%20IBM%20Watson.pdf

Agenda

- NLP terminology and low-level tasks
- Sparsity in language
- **Text pre-processing**

Text pre-processing

- The first but highly crucial step of any NLP task
  - Text pre-processing (partially) addresses the sparsity challenge, why?

We will learn:

- Text normalization
- Segmentation
- Stop words
- Stemming & Lemmatization
- Tokenization
  - Rule-based tokenization
  - Subword tokenization

Text Normalization

- Normalization harmonizes the written forms of the words with same meanings

- Some examples:
  - deleting periods
    - ***U.S.A.*** → ***USA***
  - deleting hyphens
    - ***anti-discriminatory*** → ***antidiscriminatory***
  - Accents
    - French ***résumé*** → ***resume***
  - Umlauts
    - German: ***Tuebingen*** → ***Tübingen***

Text Normalization

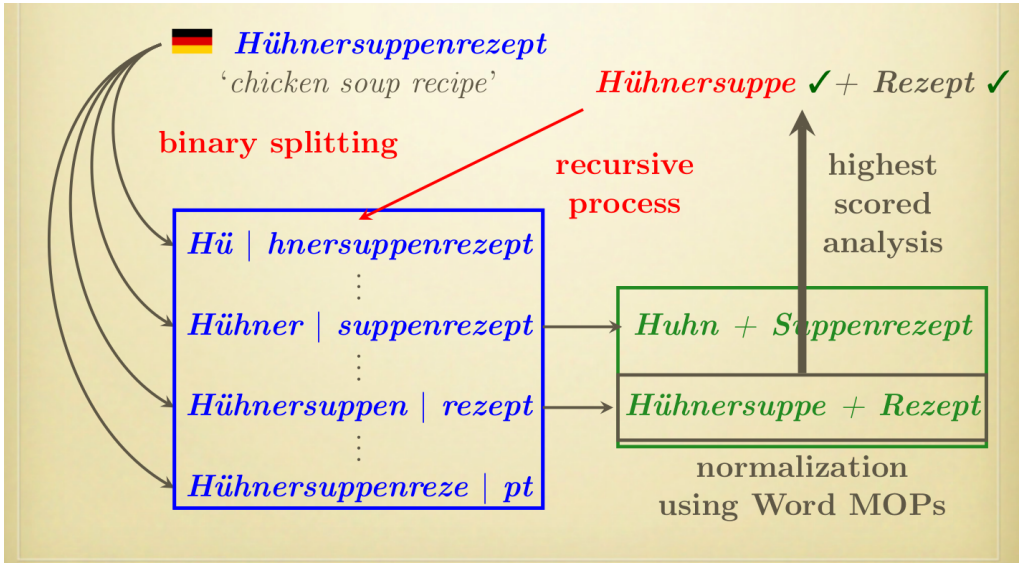
- Case folding: reduce all letters to lower case
  - It may cause ambiguity but typically helpful
    - **General Motors** vs. **general motors**
    - **Fed** vs. **fed**
    - **CAT** (City Airport Train) vs. **cat**

- Longstanding Google example:
  - Search **C.A.T.**

- Do the numbers, dates, etc. bring information?
  - If included, the dictionary size may explode!
  - Usually numbers and dates are replaced by special tokens, e.g.
    - Numbers with <num>
    - Dates with <dates>

Segmentation

- Segmentation
  - Splitting a compound word into tokens
- French
  - ***L'ensemble*** → one token or two? ***L*** ? ***L'*** ? ***Le*** ?
- German compound nouns
  - ***Halsschlagader*** → ***Hals Schlag Ader***?
  - Compound words in German usually require **compound splitter**
  - A Possible algorithm (look in the link for more details):



https://www.ims.uni-stuttgart.de/forschung/ressourcen/werkzeuge/mcs/

Stop words

- Stop words
  - The commonest words, like ***the***, ***a***, ***and***, ***to***, ***be***
  - They carry little or no semantic information

- Stop words can also be important, especially in combination with other words, e.g.:
  - Phrases: ***"King of Denmark"***, ***"To be or not to be"***
  - Titles, etc.: ***"Let it be"***
  - Definitional purposes: "**flights to London**"

- Stop words are sometimes excluded from the corpus
  - Typically in bag-of-words approaches

Stemming

- Morphemes in English consists of
  - Stem: the core meaning-bearing units
  - Affixes: pieces that adhere to stems

- A stemmer reduces words to their “stems” by crude affix chopping. Examples:
  - ***automate, automates, automatic, automation → automat***
  - ***for example compressed and compression are both accepted as equivalent to compress →***

    for exampl compress and compress ar both accept as

    equival to compress

Porter's stemmer

- Commonest algorithm for stemming English
  - consists of a set of grammatical commands

- Typical rules:
  - *sses* $\rightarrow$ *ss*
  - *ies* $\rightarrow$ *i*
  - *ational* $\rightarrow$ *ate*
  - *tional* $\rightarrow$ *tion*

Give it a try: https://text-processing.com/demo/stem/

Lemmatization

- A lemmatizer uses a knowledge resource (like WordNet) to find and replace base forms

- Lemmatzer reduces inflectional/variant forms to base forms. Examples:
  - ***am***, ***are***, ***is*** $\rightarrow$ ***be***
  - ***car***, ***cars***, ***car's***, ***cars'*** $\rightarrow$ ***car***
  - ***the boy's cars are different colors*** $\rightarrow$ ***the boy car be different color***

- Lemmatization versus Stemming:
  - Both reduce variation
  - Stemming is typically faster
  - Stemming may harm precision and increase ambiguity
  - If a given word does not exist in the knowledge resource, lemmatization may not be able to process it

Tokenization

- Tokenization
  - Splitting a running text into tokens

- Sample questions when tokenizing:
  - ***Finland's capital*** → ***Finland*** AND ***s***? ***Finlands***? ***Finland's***?
  - ***Hewlett-Packard*** → ***Hewlett*** and ***Packard*** as two tokens?
  - ***state-of-the-art*** → break up hyphenated sequence?
  - ***San Francisco*** → one token or two? ***San_Francisco*** or "***San***" and "***Francisco***"?
  - ***lowercase*** → ***lower-case***, ***lowercase***, or ***lower case***?
  - what's, don't → ***what is***, ***do not***?

Tokenization approaches

- Two general tokenization approaches
  - Rule-based tokenization
  - Subword tokenization

- Rule-based tokenization
  - Tokenization using a set of rules
  - needs language-specific knowledge
  - can become problematic in morphologically rich languages
  - common approach to tokenization
    - For instance provided in libraries like spaCy and Moses, or by using Regular Expressions

Subword tokenization

- Motivating example:
  - "***structurally***" appears rarely, however, its meaning can be inferred from "***structure***" which may appear much more often in a corpus
  - Lemmatizers and stemmers turn "***structurally***" and "***structure***" to the same stem (like "***structur***"), they both
    1. require knowledge about the specific language in hand (like English or Swahili)
    2. remove the differences between these two words

- Subword tokenization uses corpus statistics to first create a vocabulary list of subwords, and then decomposes every word to these subwords.

- Subword tokenization does not need any knowledge about language but uses the occurrence statistics, extracted from a corpus.

Subword tokenization

Byte Pair Encoding (BPE)

- The core idea of BPE comes from information theory and compression

- BPE (or in general subword tokenizers) consist of two steps:
  1. **Training:** Learning a vocabulary list of subwords from a given corpus
  2. **Tokenization (decoding):** tokenize a given text using the stored subwords vocabulary list

Reference: Sennrich, R., Haddow, B., & Birch, A.. Neural Machine Translation of Rare Words with Subword Units.
In *proceedings of ACL (2016)*

Byte Pair Encoding (BPE)

Sketch of training:

1. Start from a vocabulary list with all single characters
2. Pre-tokenize the training corpus, for instance simply by splitting on white spaces
3. Create a dictionary of words and counts from the pre-tokenized training corpus
4. Add special character "_" to the end of each word in the dictionary
5. Expand the vocabulary list:
   - Find the most frequent pair of characters in the dictionary of words
   - Merge the characters, and add them to the vocabulary list
   - Repeat step 5 till some limits on vocabulary size are reached

Learn more at "*Speech and Language Processing (3rd ed.) D. Jurafsky and J. H. Martin*", section 2.4.3.
https://web.stanford.edu/~jurafsky/slp3/2.pdf (the resource of the example)

Byte Pair Encoding – example

- Consider a tiny training corpus that leads to the following dictionary and vocabulary list

	dictionary	**vocabulary**
5	l o w _	_, d, e, i, l, n, o, r, s, t, w
2	l o w e s t _	
6	n e w e r _	
3	w i d e r _	
2	n e w _	

	dictionary	vocabulary
5	l o w _	_, d, e, i, l, n, o, r, s, t, w
2	l o w e s t _	
6	n e w e r _	
3	w i d e r _	
2	n e w _	

First merge

	dictionary	vocabulary
5	l o w _	_, d, e, i, l, n, o, r, s, t, w, r_
2	l o w e s t _	
6	n e w e r_	
3	w i d e r_	
2	n e w _	

Next merge

	dictionary	vocabulary
5	l o w _	_, d, e, i, l, n, o, r, s, t, w, r_, er_
2	l o w e s t _	
6	n e w er_	
3	w i d er_	
2	n e w _	

	dictionary	vocabulary
5	l o w _	_, d, e, i, l, n, o, r, s, t, w, r_, er_
2	l o w e s t _	
6	n e w er_	
3	w i d er_	
2	n e w _	

Next merge

	dictionary	vocabulary
5	l o w _	_, d, e, i, l, n, o, r, s, t, w, r_, er_, ew
2	l o w e s t _	
6	n ew er_	
3	w i d er_	
2	n ew _	

If we continue

Merge	Current Vocabulary
(n, ew)	_, d, e, i, l, n, o, r, s, t, w, r_, er_, ew, new
(l, o'	_, d, e, i, l, n, o, r, s, t, w, r_, er_, ew, new, lo
(lo, w)	_, d, e, i, l, n, o, r, s, t, w, r_, er_, ew, new, lo, low
(new, er_)	_, d, e, i, l, n, o, r, s, t, w, r_, er_, ew, new, lo, low, newer_
(low, _)	_, d, e, i, l, n, o, r, s, t, w, r_, er_, ew, new, lo, low, newer_, low_

WordPiece tokenization

- WordPiece is a descendent of BPE and has the following differences:

- Selecting character pairs for merging in WordPiece is based on "*minimizing the language model likelihood of the training data*"*

- WordPiece indicates internal subwords with "*##*" special symbol
  - E.g. "***unavoidable***" → ["***un***", "***##avoid***", "***##able***"]

* For more details look into the original paper:
Schuster, M., & Nakajima, K. (2012). Japanese and Korean voice search.
In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*

WordPiece tokenization

- Tokenization (decoding) is done using MaxMatch algorithm
  - A greedy longest-match-first algorithm
  - MaxMatch choses the longest token in the vocabulary that matches the given word
  - After a match, it repeats the previous step with the remainder of the word

```
function MAXMATCH(string, dictionary) returns list of tokens T

    if string is empty
        return empty list
    for i ← length(sentence) downto 1
        firstword = first i chars of sentence
        remainder = rest of sentence
        if InDictionary(firstword, dictionary)
            return list(firstword, MaxMatch(remainder,dictionary) )
```

WordPiece tokenization

- WordPiece with MaxMatch decoding is used in BERT
  - We will see more in the workshop session.

- Example *"**natural language processing**"* →

First pre-tokenization: [*"**natural**"*, *"**language**"*, *"**processing**"*] →

Then subword tokenization: [*"**natural**"*, *"**lang**"*, *"**##uage**"*, *"**process**"*, *"**##ing**"*]

```
function MAXMATCH(string, dictionary) returns list of tokens T

    if string is empty
        return empty list
    for i ← length(sentence) downto 1
        firstword = first i chars of sentence
        remainder = rest of sentence
        if InDictionary(firstword, dictionary)
            return list(firstword, MaxMatch(remainder,dictionary) )
```

Text pre-processing

What we have learned:

- Text normalization
- Segmentation
- Stop words
- Stemming & Lemmatization
- Tokenization
  - Rule-based tokenization
  - Subword tokenization: BPE and WordPiece

NLP Libraries

AllenNLP

gensim

Stanford CoreNLP

spaCy

huggingface.co

polyglot

PYTORCH