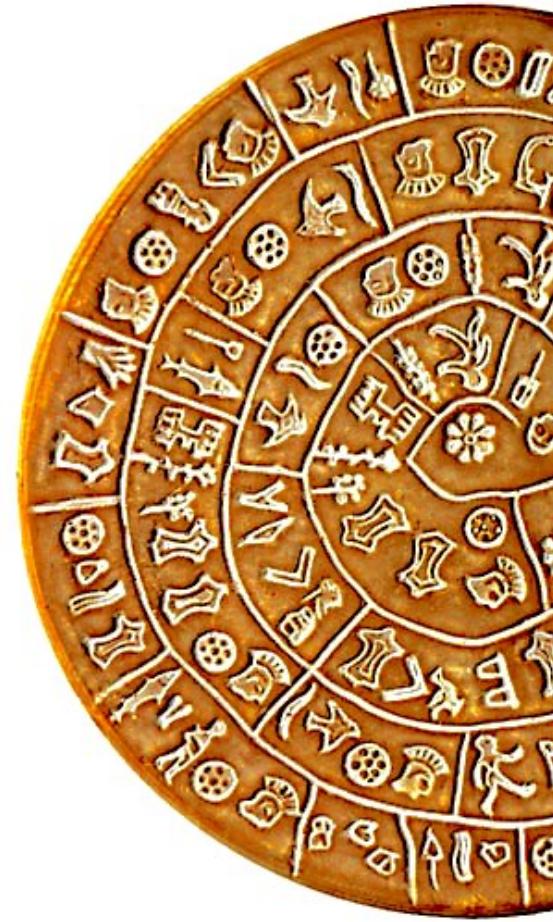


Natural Language Processing

1: Introduction





Objectives

to be able to answer questions

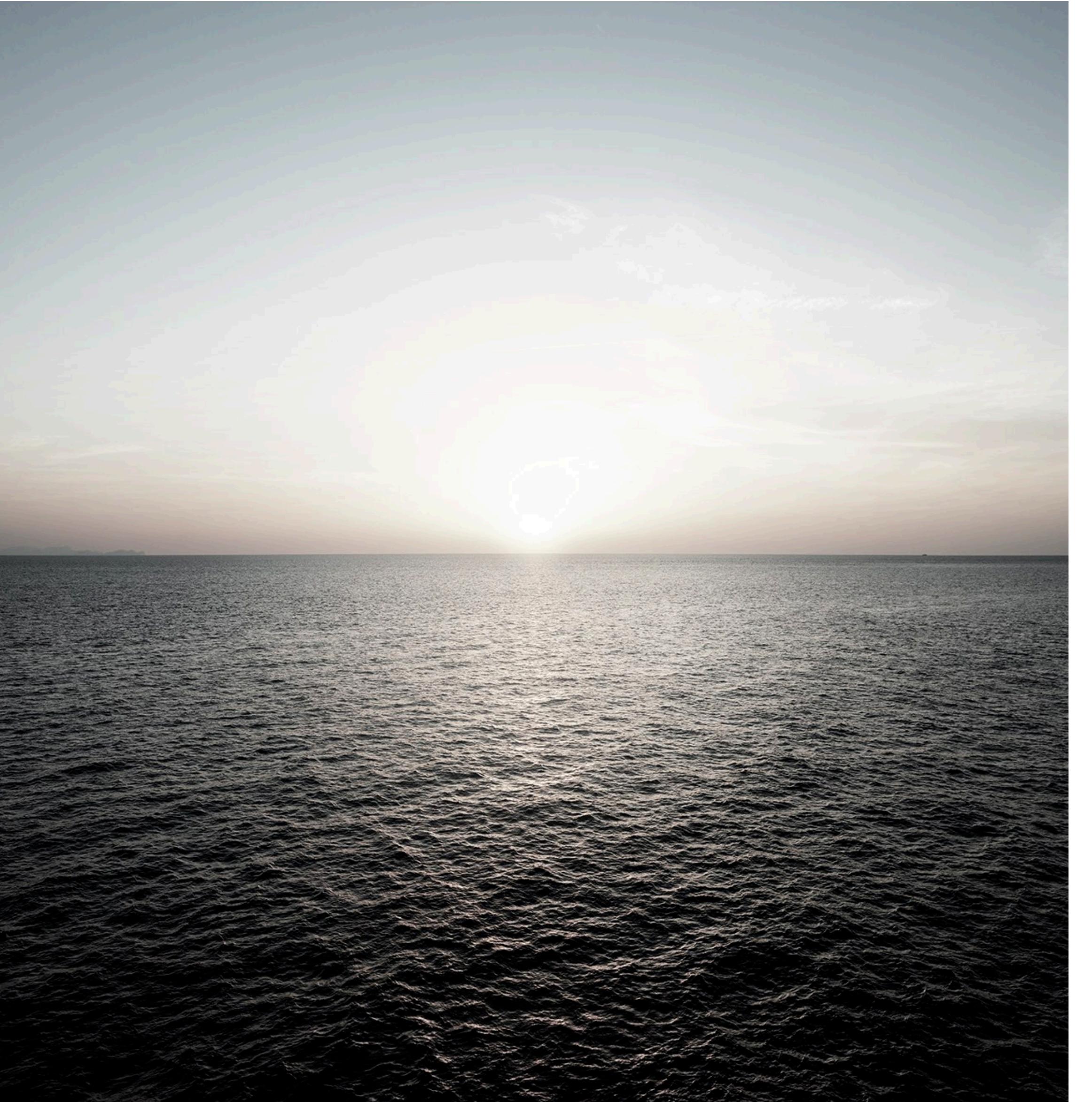
- (1) What is Natural Language Processing (NLP)?
- (2) What are common tasks in NLP?
- (3) What are the challenges in NLP?
- (4) What techniques are used in NLP?
- (5) What are typical applications of NLP?

What is Natural Language Processing (NLP)?

NLP

field of study

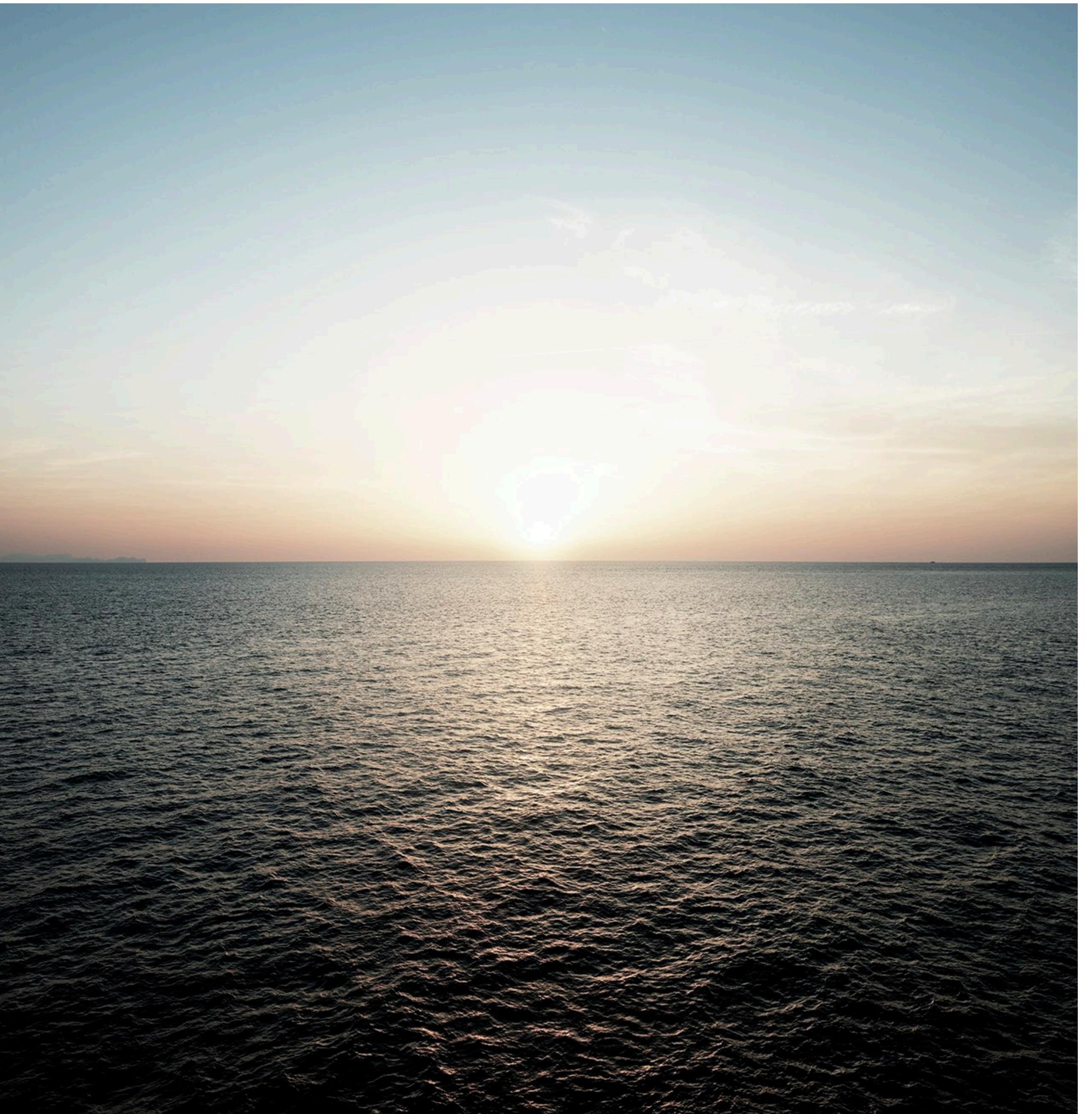
- (1) is a subfield of Artificial Intelligence
- (2) combines
 - (1) Linguistics
 - (2) Computer Science
 - (3) Machine Learning
- (3) makes human language intelligible to computers



NLP

goals

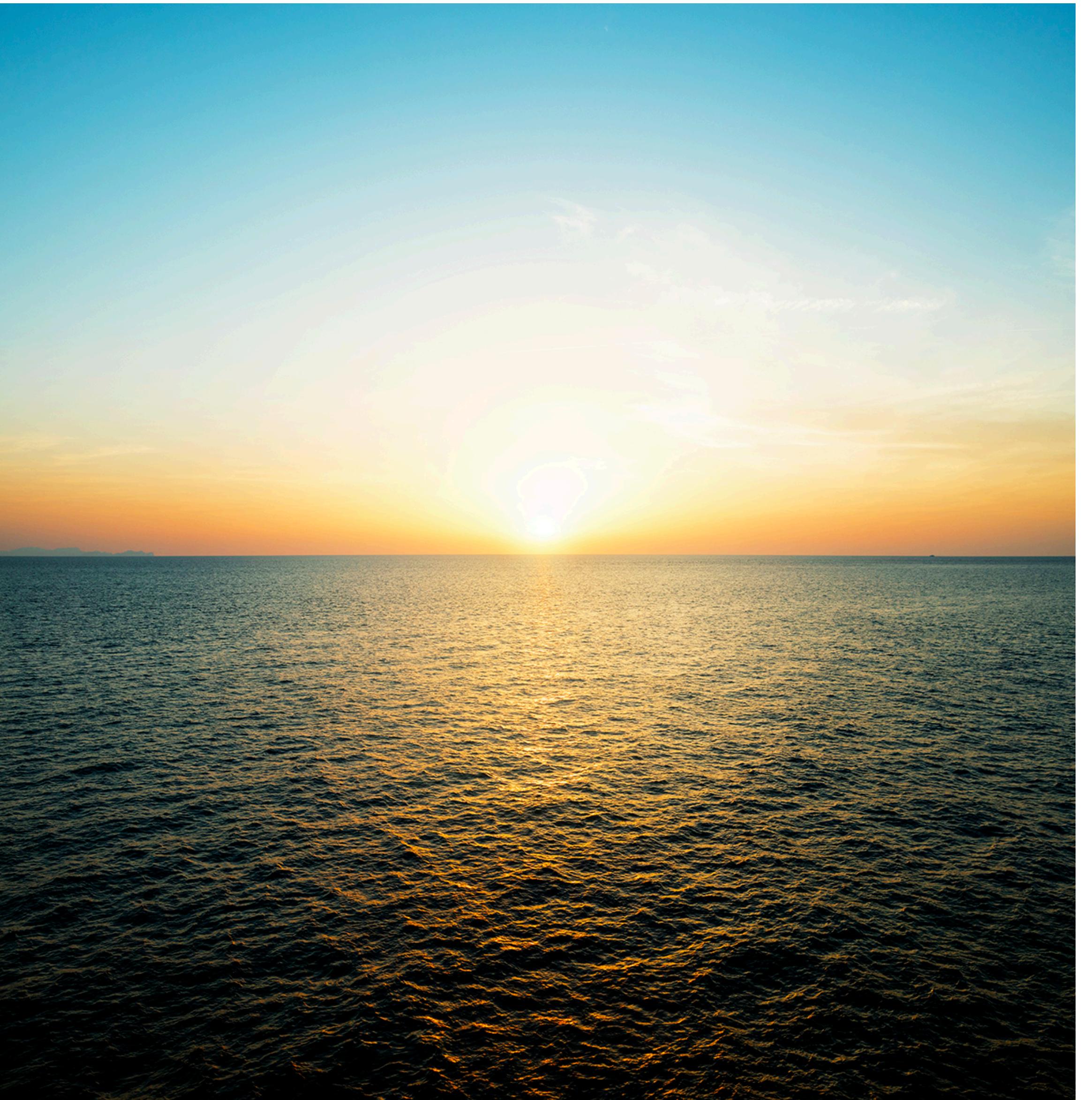
- (1) understanding the structure and meaning of human language
- (2) develop algorithms / models
- (3) create computer systems capable of
 - (1) understanding
 - (2) analyzing
 - (3) extracting meaning from text and speech



NLP

algorithms / models for

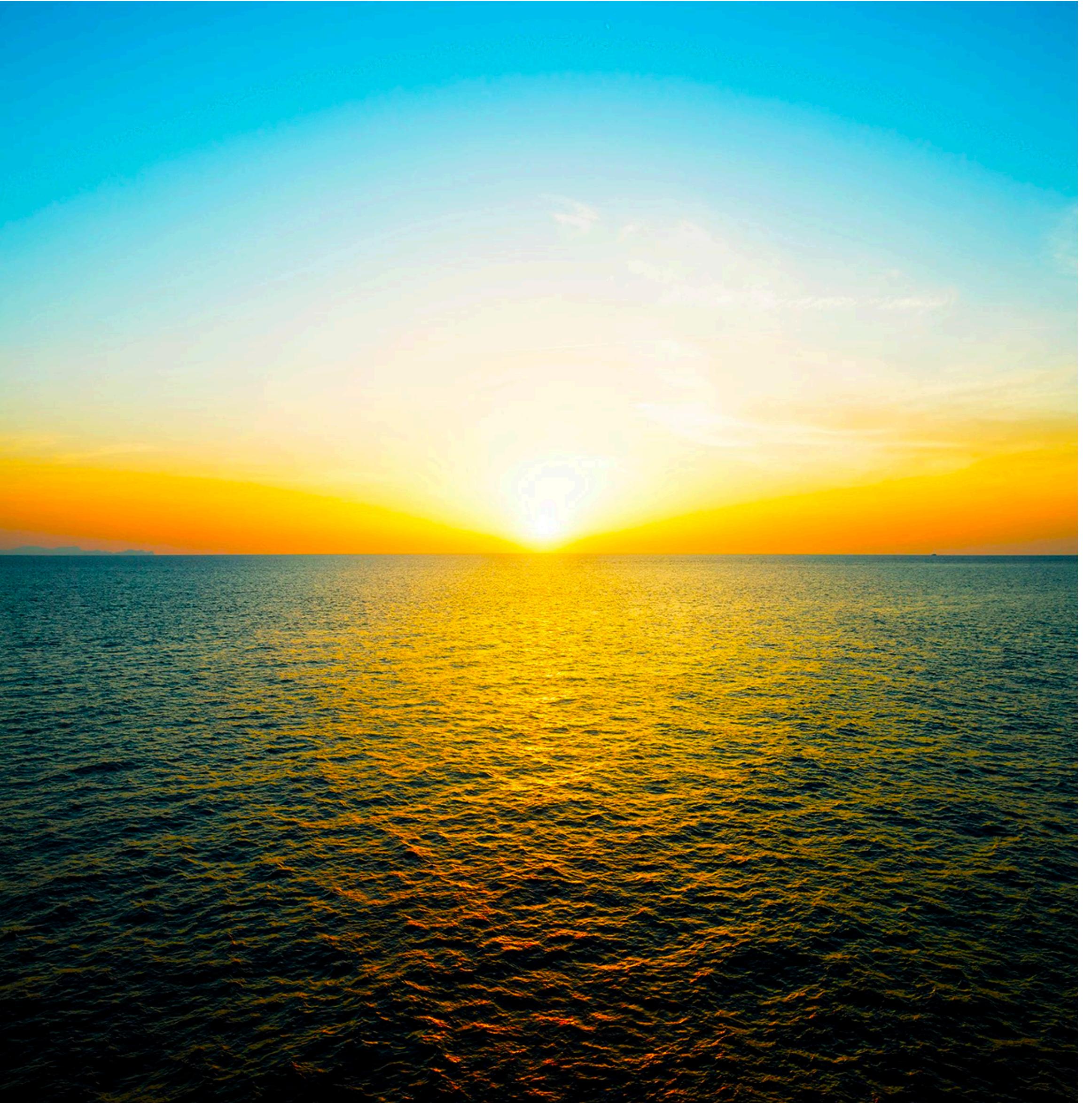
- (1) analysis of linguistic phenomena:
 - (1) morphology,
 - (2) syntax,
 - (3) semantics, and
 - (4) pragmatics
- (2) automatic performing specific tasks to solve practical problems



NLP

benefits

- (1) Large-scale text / speech analysis
- (2) Automate business processes
- (3) Common pipelines / tools can be tailored to specific application or industry (e.g. in Software Eng.)
- (4) Studies of Cognition, Perception, Human Brain, ...



**What are the challenges in
NLP?**

Main challenges

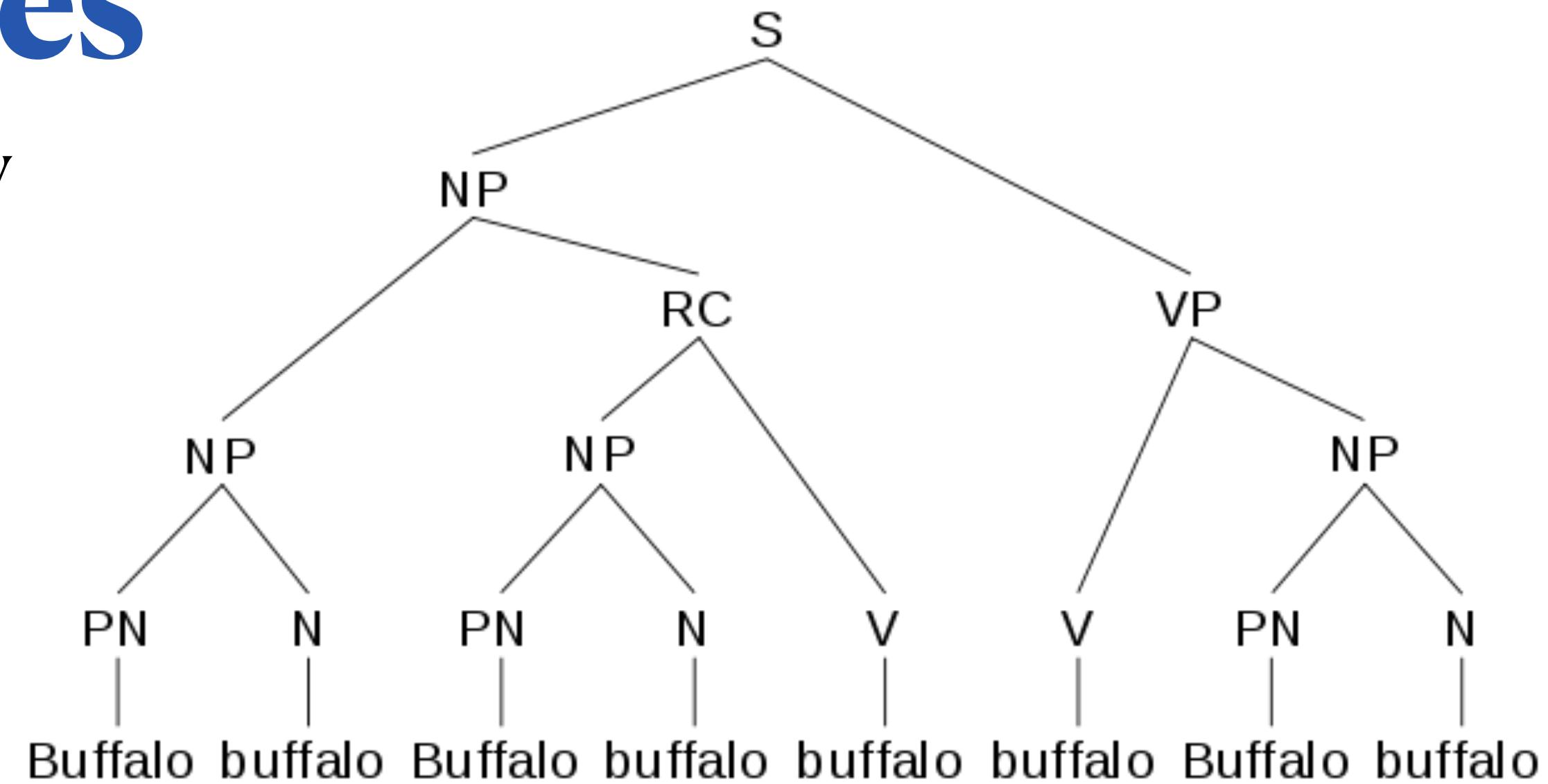
one word: “ambiguity”

- (1) Ambiguity is an intrinsic feature of human language
 - (1) meaning depends on context
 - (2) you cannot avoid it
 - (3) ambiguity presents at all levels (syntax, semantics, etc.)
 - (4) tradeoff between complexity and expressiveness
- (2) Infinitive number of possible sentences in ANY natural language
- (3) Permanent changes of natural languages / variability

Examples of ambiguity

- (1) One morning I shot an elephant in my pajamas.
How he got in my pajamas, I don't know.
-- Groucho Marx

- (2) A: What has happened to the roast beef?
B: The dog is looking very happy.



[https://en.wikipedia.org/wiki/
Buffalo_buffalo_Buffalo_buffalo_buffalo_bu
ffalo_Buffalo_buffalo](https://en.wikipedia.org/wiki/Buffalo_buffalo_Buffalo_buffalo_buffalo_buffalo_Buffalo_buffalo)

**What are common tasks /
techniques in NLP?**

Tasks, aka Problems

Two large tasks

(1) Syntactic analysis

(1) Understanding structure

(2) Semantic analysis

(1) Understanding meaning

Subtasks

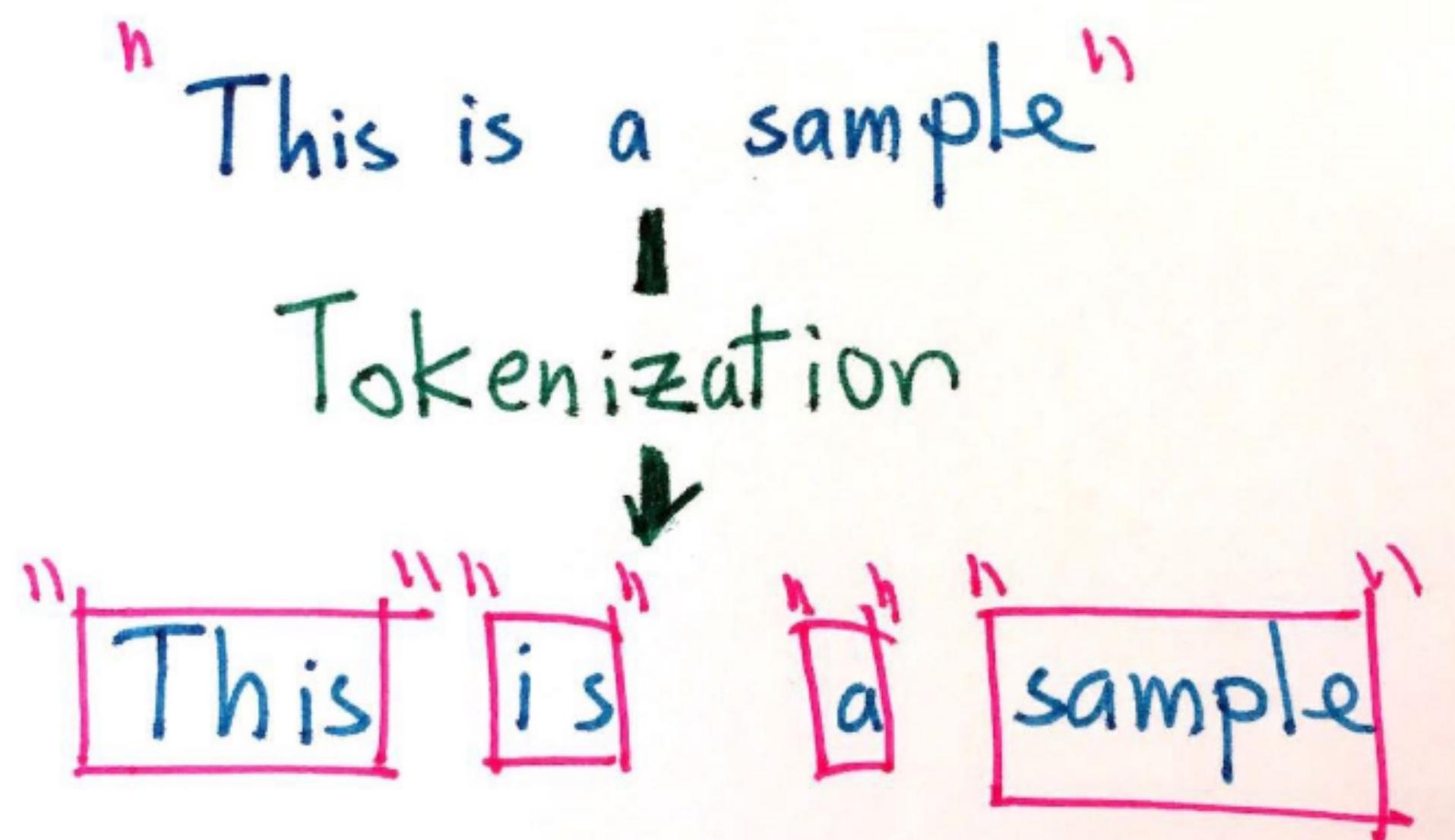
many subtasks

- (1) Tokenization, Lemmatization, Stemming
- (2) Part of Speech Tagging (PoS)
- (3) Parsing: Dependency Parsing, Constituency Parsing
- (4) Semantic Role Labeling
- (5) Word Sense Disambiguation
- (6) Information Extraction:
 - (1) Named Entity Recognition (NER)
 - (2) Relation / Event Extraction
- (7) Entity Linking

**Tokenization,
Lemmatization, Stemming**

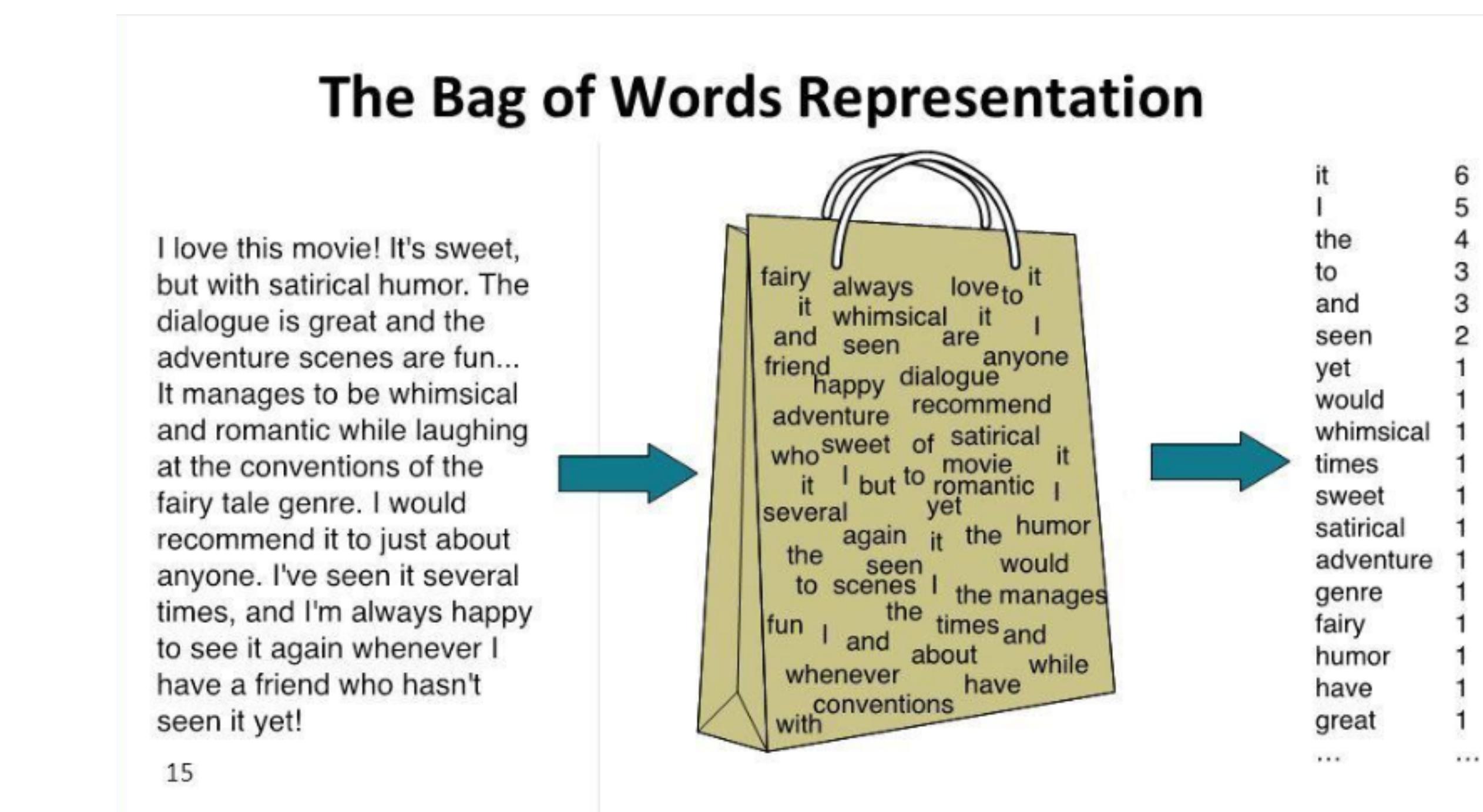
Tokenization

- (1) Breaking the input text into meaningful elements, smaller pieces or "tokens".
- (2) Tokens can be:
 - (1) elements for further semantic processing
 - (2) word, sentence, paragraph
- (3) Certain characters, such as punctuation marks, can be removed at the same time.



Bag of words (BoW)

- (1) Machine learning algorithms cannot operate on raw text directly; text must be converted to numbers (more precisely, to vectors of numbers)
- (2) BoW: representing text data when modeling text with ML
- (3) Feature extraction or encoding.
- (4) Ignoring the order of words in the text



BPE

Byte Pair Encoding

- (1) encoding (rare) words via subword units
- (2) eliminating the need for large vocabularies
- (3) subword models achieve better accuracy for the translation of rare words than large-vocabulary models

Algorithm 1 Learn BPE operations

```
import re, collections

def get_stats(vocab):
    pairs = collections.defaultdict(int)
    for word, freq in vocab.items():
        symbols = word.split()
        for i in range(len(symbols)-1):
            pairs[symbols[i],symbols[i+1]] += freq
    return pairs

def merge_vocab(pair, v_in):
    v_out = {}
    bigram = re.escape(' '.join(pair))
    p = re.compile(r'(?<!\\S)' + bigram + r'(?!\\S)')
    for word in v_in:
        w_out = p.sub(' '.join(pair), word)
        v_out[w_out] = v_in[word]
    return v_out

vocab = {'l o w </w>' : 5, 'l o w e r </w>' : 2,
         'n e w e s t </w>':6, 'w i d e s t </w>':3}
num_merges = 10
for i in range(num_merges):
    pairs = get_stats(vocab)
    best = max(pairs, key=pairs.get)
    vocab = merge_vocab(best, vocab)
    print(best)
```

Sparse vs. Dense Encodings

Discussion

(1) TF*IDF

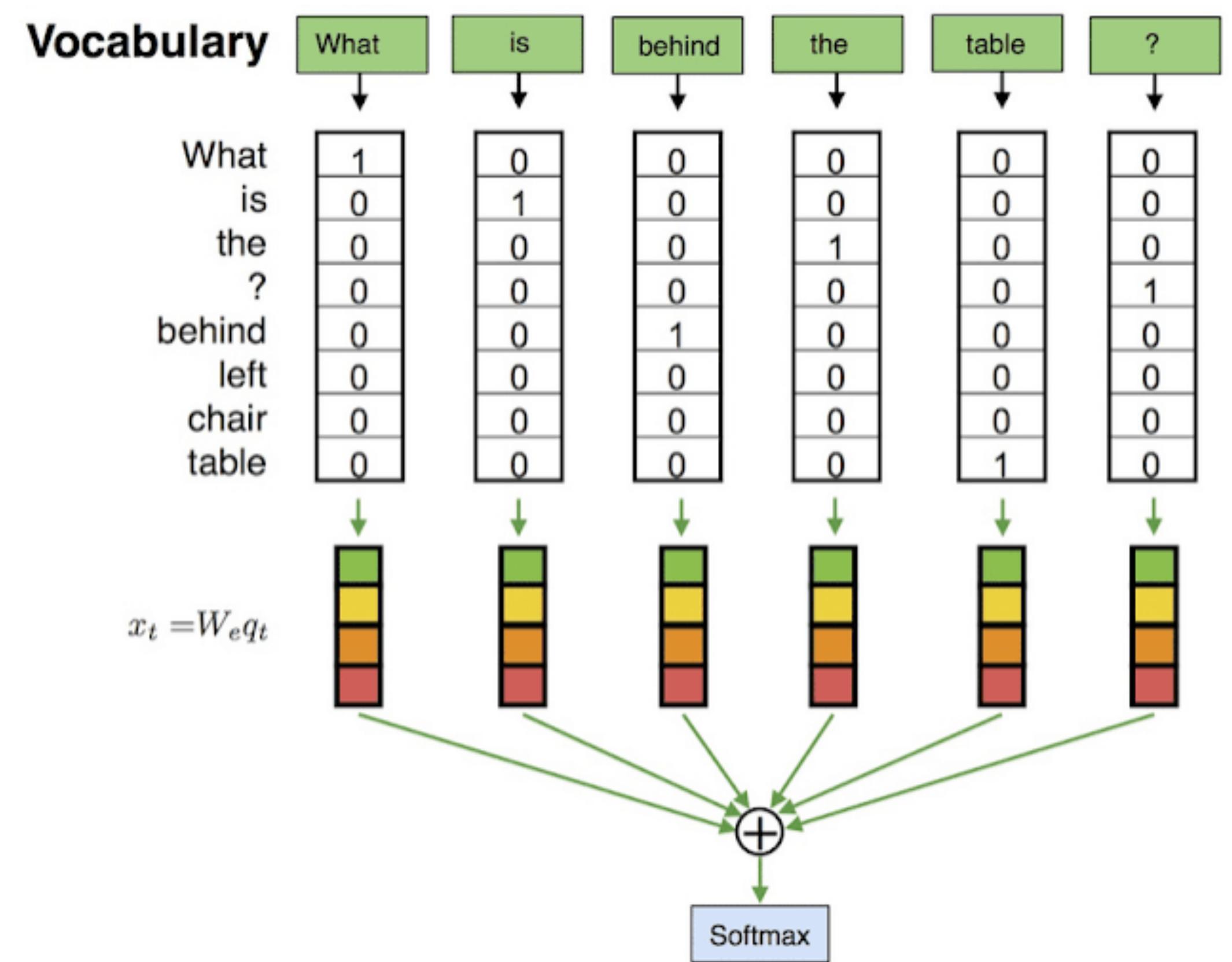
(1) can you recall a formula, please?

(2) Dense vectors:

(1) Word2Vec, GloVe

(2) ELMo

(3) BERT,...



What are typical
applications of NLP?

Examples of applications

only few

- (1) Text classification (email spam filters)
- (2) Virtual assistants
- (3) Search engines
- (4) Autocorrect, grammar checkers
- (5) Social media analysis (sentiment, trends, argument mining)
- (6) Customer feedback, customer support
- (7) Chatbots
- (8) Machine translation
- (9) Natural language generation
- (10) Automatic text summarization, simplification

Text Classification

(1) Binary: spam / non-spam

(2) Multi-class

(1) sport / politics / economics

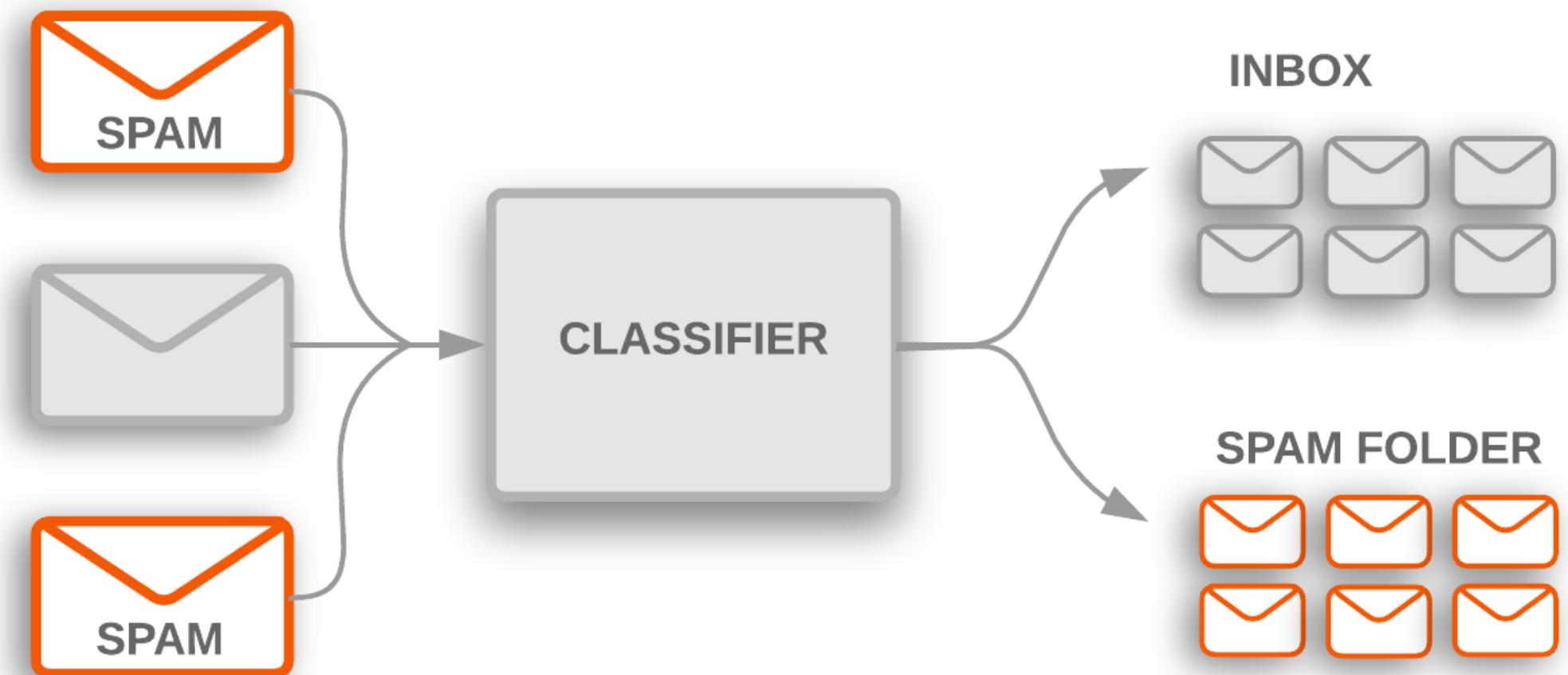
(2) hierarchical

(3) Multi-label

(1) toxic texts detection

(2) propaganda classification

(4) Topic modeling



Technique	Train
Appeal to Authority	144
Appeal to Fear-Prejudice	294
Bandwagon/Reductio ad Hitlerum	72
Black and White Fallacy	107
Causal Oversimplification	209
Doubt	493
Exaggeration/Minimisation	466
Flag Waving	229
Loaded Language	2123
Name Calling/Labeling	1058
Repetition	621
Slogans	129
Thought-Terminating Cliches	76
Whataboutism/Straw Men/Red Herring	108

Virtual assistants

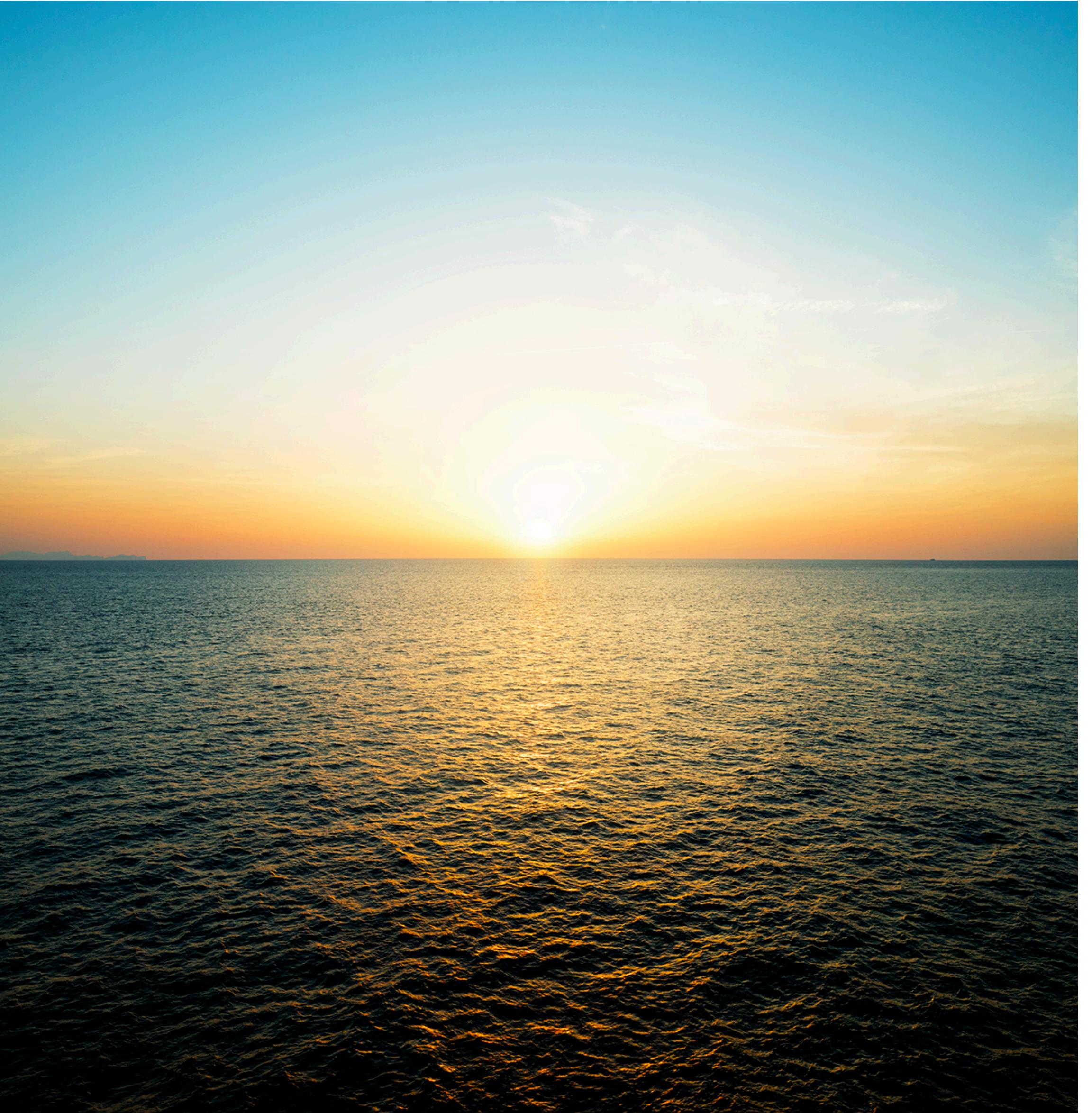
Let them talk!

- (1) Alexa, Alisa, Cortana, Google, Siri, ...
- (2) Processing voice requests
 - (1) speech to text
 - (2) recall previous interactions
 - (3) connect to apps
 - (4) control devices

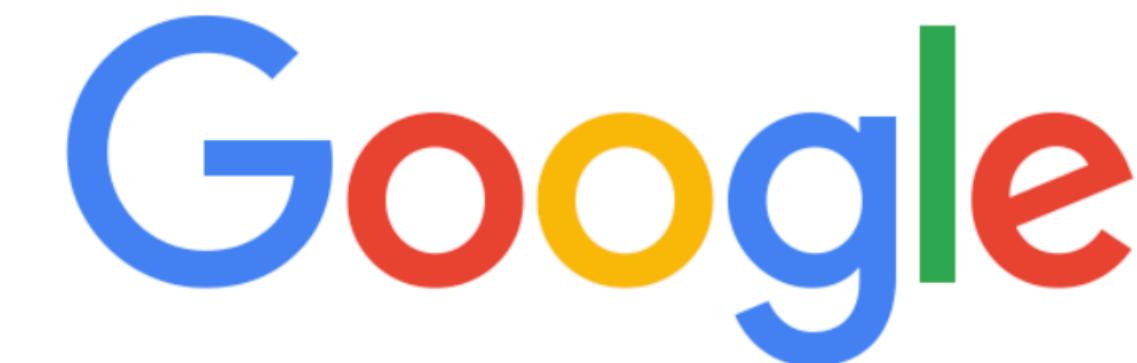


Search engines

- (1) Processing of documents and queries
 - (1) clear content
 - (2) autocomplete
 - (3) query expansion
- (2) suggesting queries / topics
- (3) ...

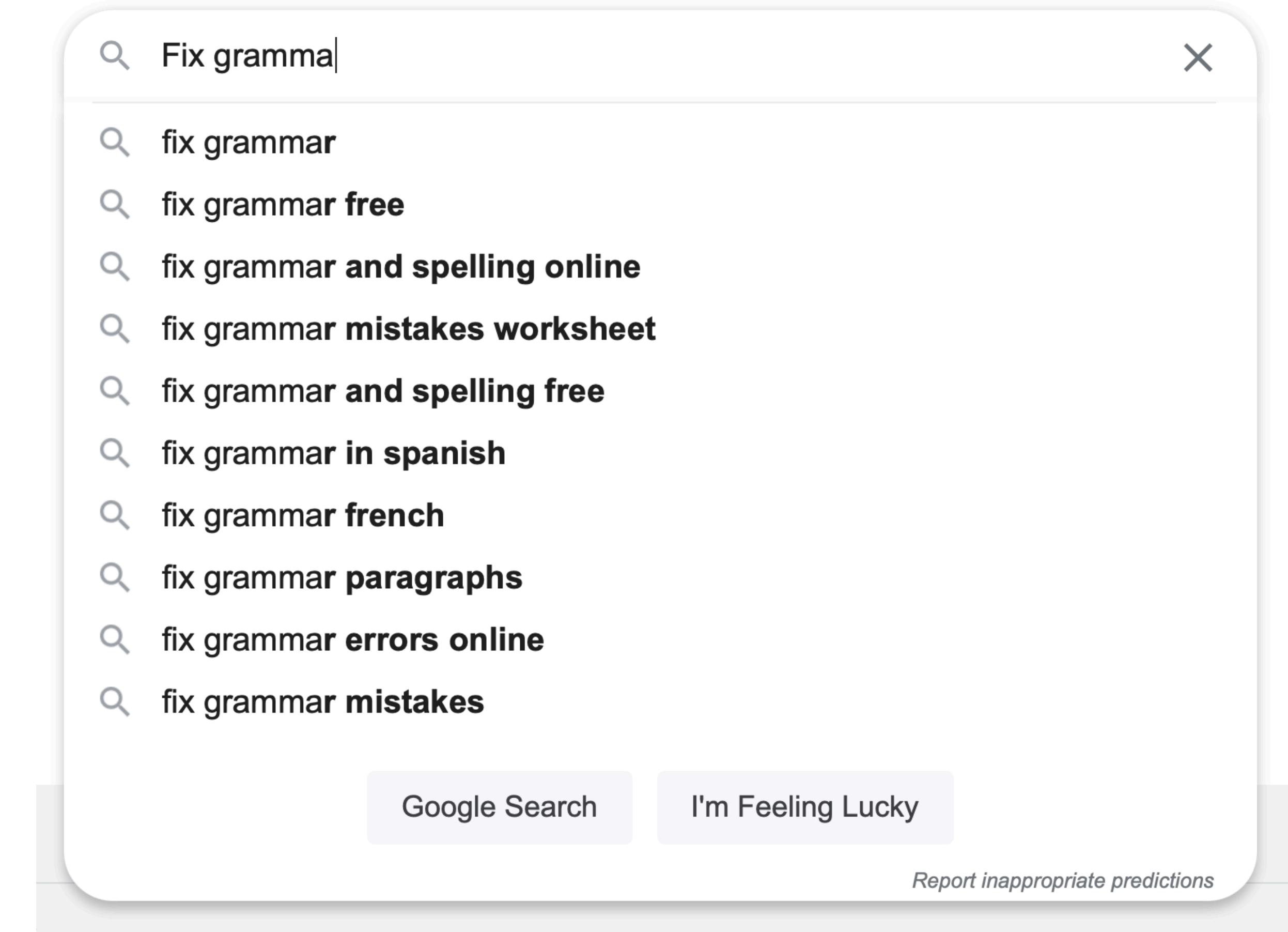


Autocorrect, grammar checkers



- (1) Predicts text as you type it
- (2) Fix grammar **miskates**

go back to grammar school, organic entity



A screenshot of a Google search bar showing suggestions for "Fix grammar". The suggestions include:

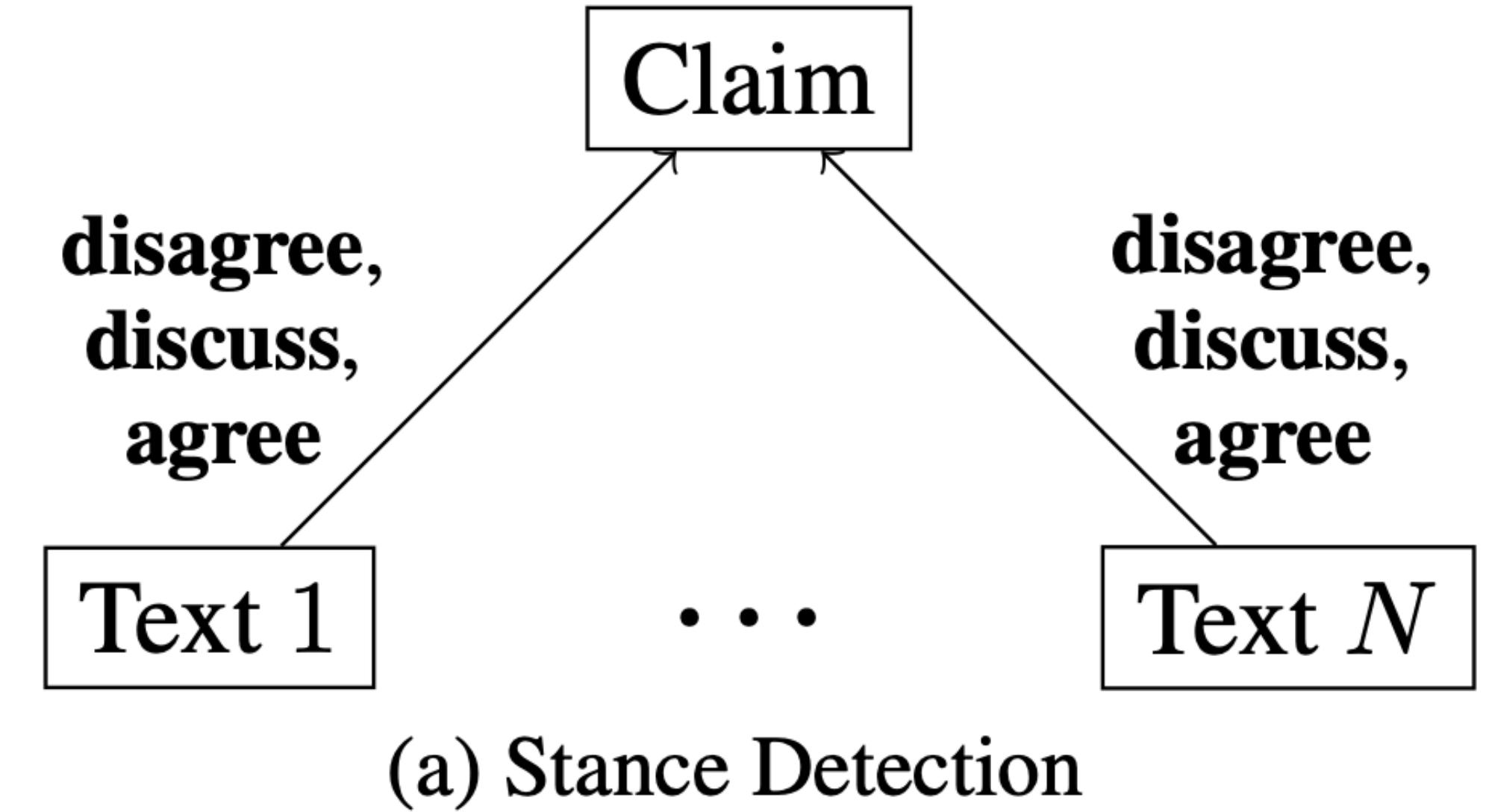
- Fix grammar
- fix grammar
- fix grammar free
- fix grammar and spelling online
- fix grammar mistakes worksheet
- fix grammar and spelling free
- fix grammar in spanish
- fix grammar french
- fix grammar paragraphs
- fix grammar errors online
- fix grammar mistakes

Below the suggestions are two buttons: "Google Search" and "I'm Feeling Lucky". A small link "Report inappropriate predictions" is located at the bottom right of the search bar.

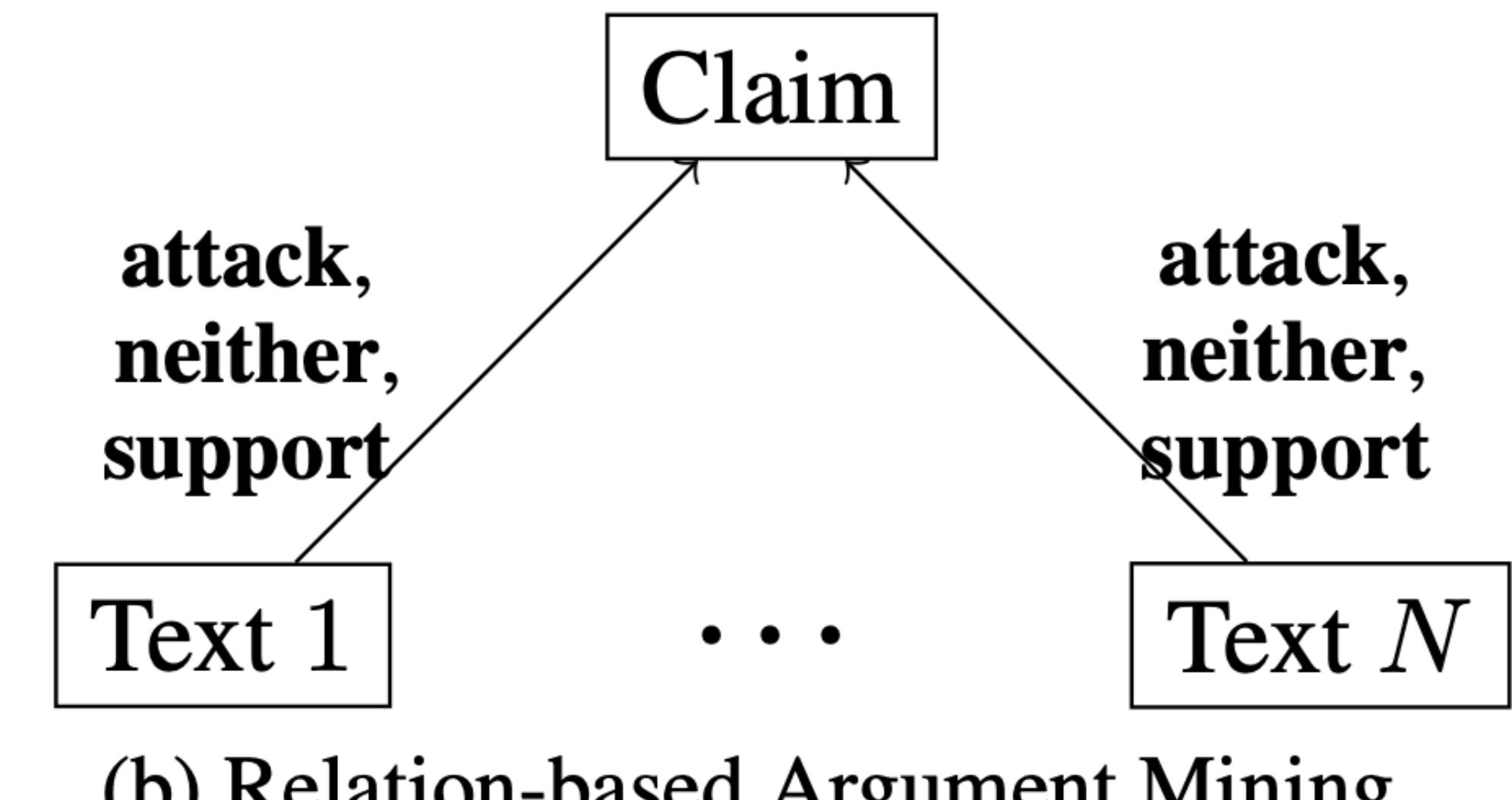
Social media analysis

Monitoring society

- (1) Sentiment analysis
- (2) Aspect-based sentiment
- (3) Opinion mining
- (4) Argument mining: classifying arguments and relationships between them
- (5) Brand tracking



(a) Stance Detection

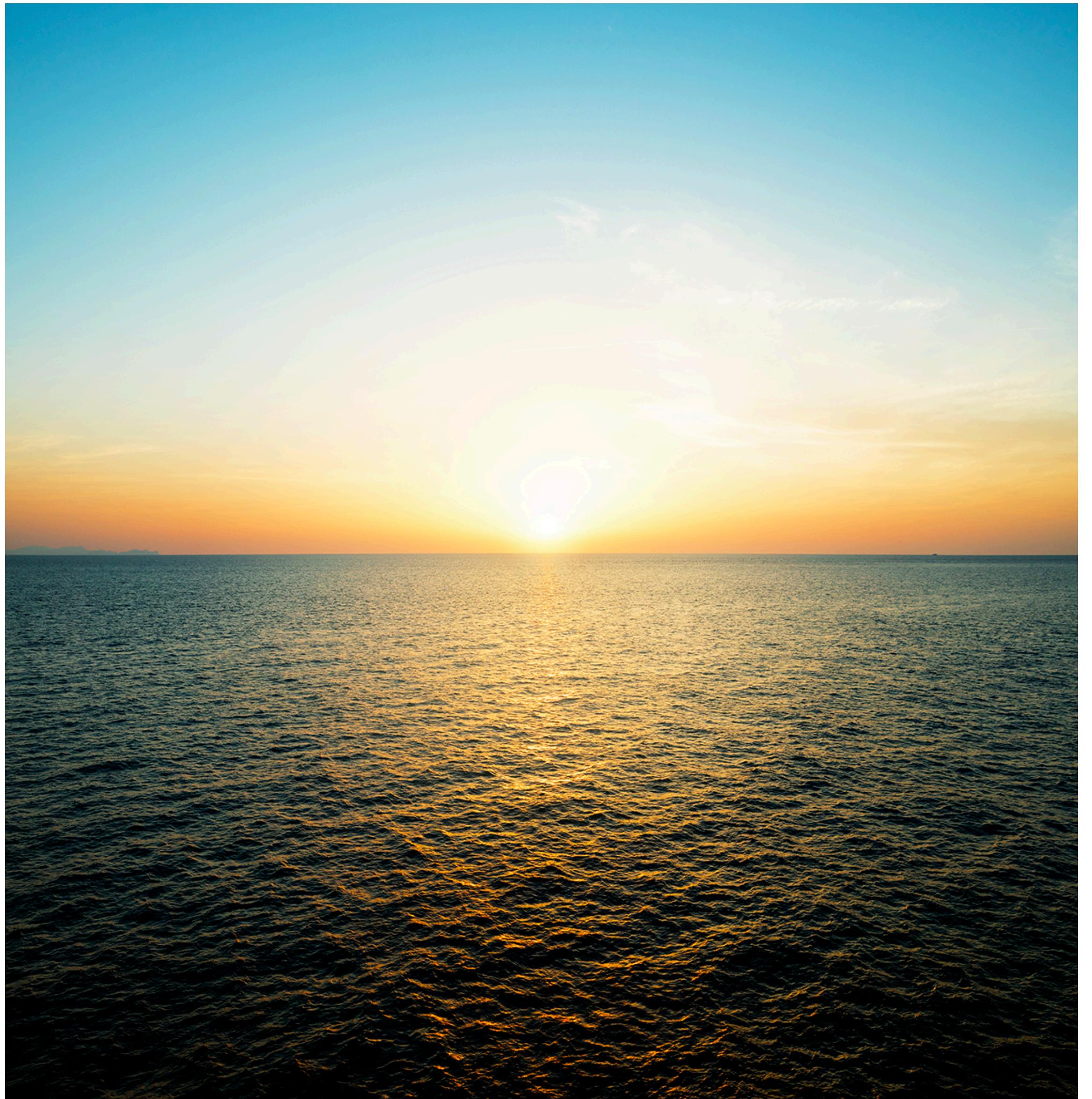


(b) Relation-based Argument Mining

Customer feedback

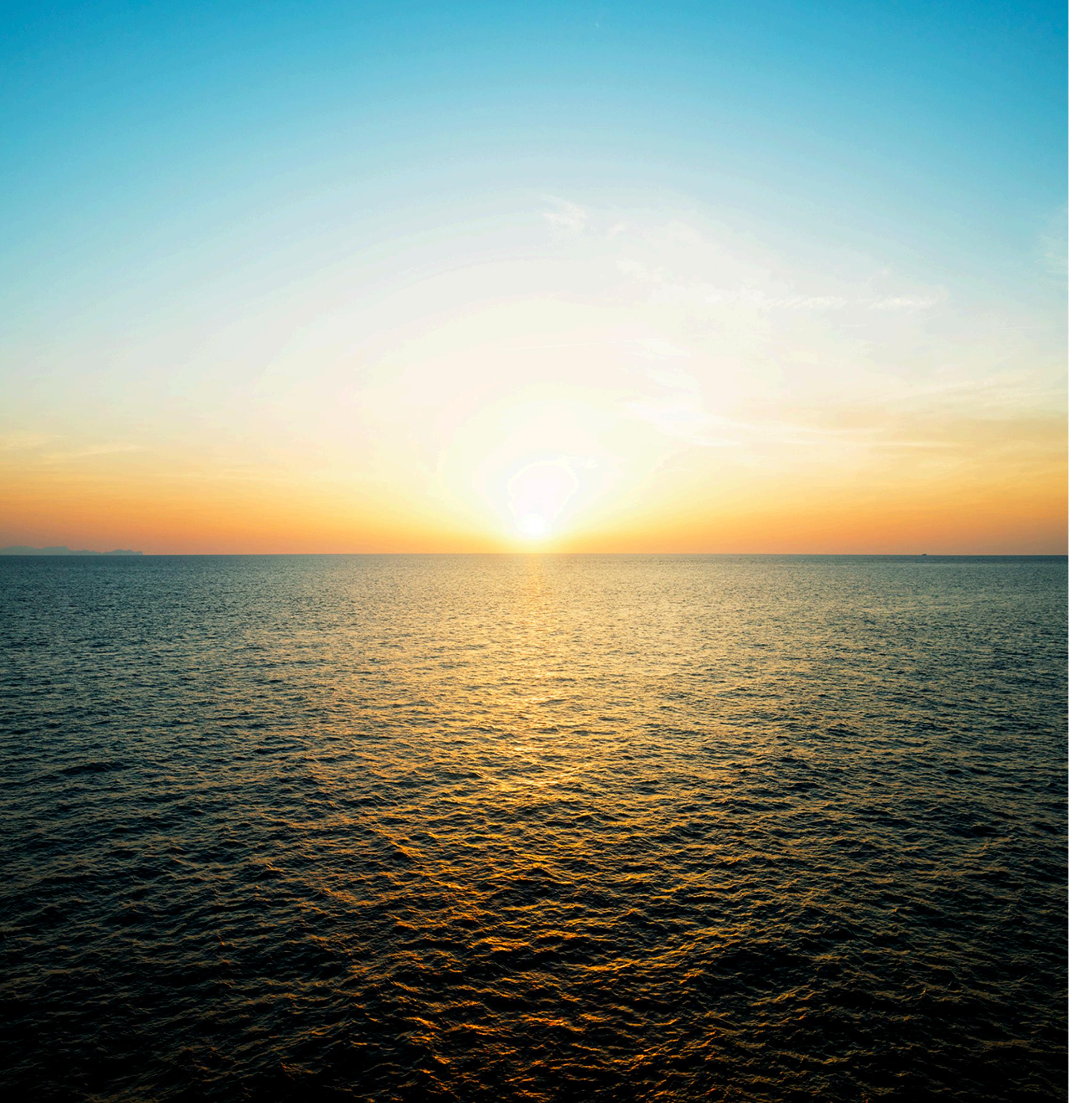
Customer analysis

- (1) User Feedback analysis
 - (1) Topics mentioned
 - (2) Response classification
 - (3) Rounting user questions
 - (4) Tickets in customer support
- (2) Question answering
 - (1) about products
 - (2) about company / processes



Chatbots

- (1) Simulates human conversation
- (2) Intent and emotion recognition
- (3) Auxiliary tasks:
 - (1) recommendation
 - (2) offer discounts
 - (3) make reservations
- (4) usually, simple tasks: if/then...



Machine translation

since 1950s

(1) Milestones in MT:

- (1) rule-based
- (2) statistical MT
- (3) neural MT

(2) In 2019 Facebook AI (FAIR):

- (1) English-German, English-Russian
- (2) “superhuman” performance
- (3) better than previous system by
4.5 BLEU

Key features of the MT model:

- cross-lingual pretraining and
- self-supervised learning

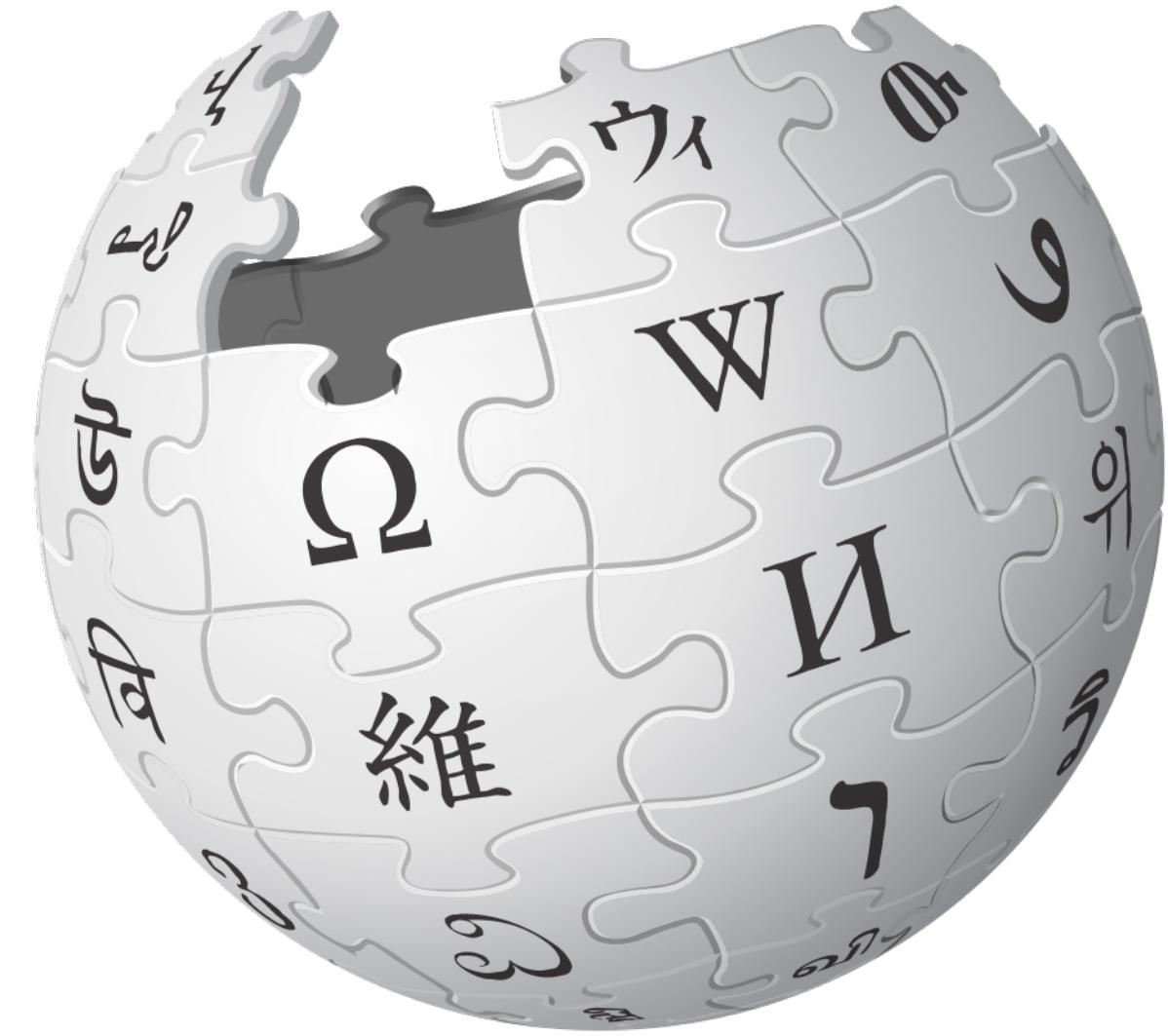
<https://ai.facebook.com/blog/facebook-leads-wmt-translation-competition/>

<https://aclanthology.org/W19-5333.pdf>

Automatic text summarization

keep it simple...

- (1) Extractive
- (2) Abstractive
- (3) Text simplification



Simple English
WIKIPEDIA

Natural language generation

<https://transformer.huggingface.co/doc/distil-gpt2>

Prompt is All you need (plus Attention)

- (1) Automatic generation of different texts
- (2) GPT-2 breakthrough (in 2019)
- (3) Generating whole stories (e.g. about unicorns)
- (4) Fake news articles
- (5) Yes, there is GPT-3...
- (6) Yes, there is ChatGPT...



Write With Transformer

Get a modern neural network to auto-complete your thoughts.

natural language generation is
a good tool to understand language development.

Post-test

Summary

Finding more about NLP

Hunt for NLP nuggets

(1) Find 1-3 things related to and not covered in slides:

(1) Challenges, Models, Applications and NLP tools

(2) Write a report:

(1) What have you found?

(2) Why is it important?

(3) Send to the course chat: 1-2 sentences + links

(4) (Optional) Vote as reactions

Course Q/A Syllabus Review Grading

Syllabus and grading

https://eduwiki.innopolis.university/index.php/BSc:_Natural_Language_Processing

Course grading range

Grade	Range	Description of performance
A. Excellent	90-100	-
B. Good	75-89	-
C. Satisfactory	60-74	-
D. Poor	0-59	-

Course activities and grading breakdown

Activity Type	Percentage of the overall course grade
Final Exam	30
Final project	30
Assignments	30
Lab Participation / Quizzes	10

Week	Date	Topic
1	25.01.2023	Introduction to NLP, Computational Linguistics. Problems, pipelines
2	01.02.2023	Text preprocessing. Tokenization. Sentence Segmentation
3	08.02.2023	Lexical analysis. POS-tagging
4	15.02.2023	Parsing. Dependency parsing
5	22.02.2023	Language modelling (Ngram model)
6	01.03.2023	Semantic Analysis. WordNets
7	08.03.2023	Distributional semantics (Word2vec, fastText)
8	15.03.2023	Midterm
9	22.03.2023	Neural models in NLP
10	29.03.2023	CNNs for text classification
11	05.04.2023	RNNs in NLP
12	12.04.2023	RNNs in NLP
13	19.04.2023	Transfer learning in NLP
14	26.04.2023	Transformer-based architectures
15	03.05.2023	Extra week. Project defences
16	10.05.2023	Final Exam

Books

- (1) **(SLP)** Speech and Language Processing (3rd ed. draft). Dan Jurafsky and James H. Martin.
<https://web.stanford.edu/~jurafsky/slp3/>
- (2) **(HNLP)** Handbook Of Natural Language Processing, Second Edition Chapman & Hall Crc
Machine Learning & Pattern Recognition 2010
- (3) **(HoPNLP)** Kedia, Aman; Rasu, Mayank. Hands-On Python Natural Language Processing:
Explore tools and techniques to analyze and process text with a view to building real-world
NLP applications (p. 153). Packt Publishing.
- (4) **(HOML)** Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow. Aurélien
Géron

Natural Language Processing

2: Tokenization and Segmentation

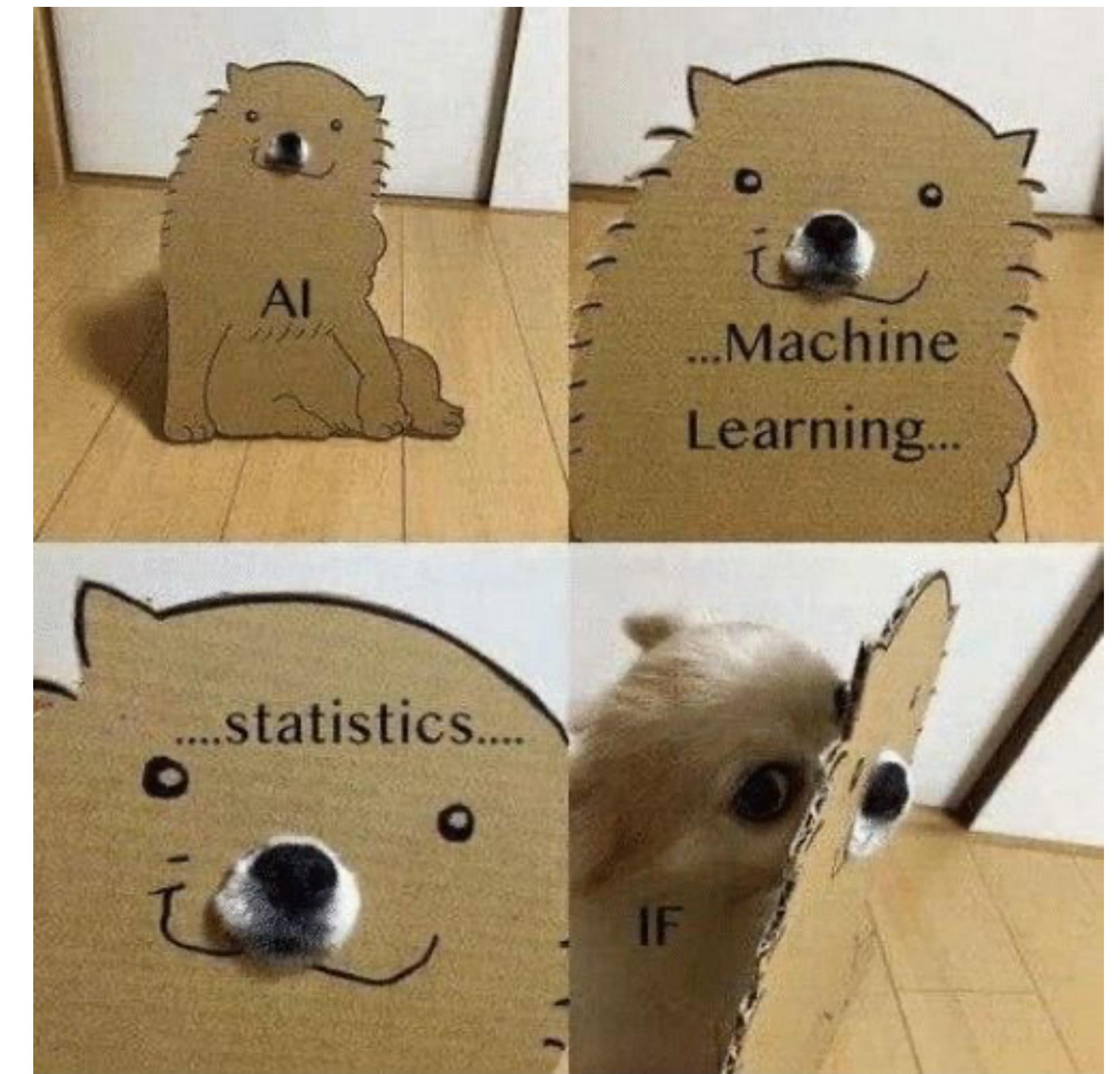




Objectives

to be able to answer questions

- (1) What methods exist for Tokenization and Segmentation?
- (2) What is the place of a tokenizer in NLP pipeline?
- (3) How does BPE work?



Words and Corpora

How many words in a sentence?

(1)"I do uh main- mainly business data processing"

- Fragments, filled pauses

(2)"Seuss's **cat** in the hat is different from other **cats!**"

- Lemma: same stem, part of speech, rough word sense
 - **cat** and **cats** = same lemma
- Wordform: the full inflected surface form
 - **cat** and **cats** = different wordforms

How many words in a sentence?

They lay back on the San Francisco grass and looked at the stars and their

- Type: an element of the vocabulary
- Token: an instance of that type in running text
- How many?
 - 15 tokens (or 14)
 - 13 types (or 12) (or 11?)

How many words in a corpus?

N = number of tokens

V = vocabulary = set of types, $|V|$ is size of vocabulary

Heaps Law = Herdan's Law = $|V| = kN^\beta$ where often $.67 < \beta < .75$

i.e., vocabulary size grows with $>$ square root of the number of word tokens

Corpus	Tokens = N	Types = $ V $
Switchboard phone conversations	2.4 million	20 thousand
Shakespeare	884,000	31 thousand
COCA	440 million	2 million
Google N-grams	1 trillion	13+ million
Common Crawl / mC4	6.3 trillion	??

Corpora

corpora = corpuses

Words don't appear out of nowhere!

A text is produced by

- (1) a specific writer(s),
- (2) at a specific time,
- (3) in a specific variety,
- (4) of a specific language,
- (5) for a specific function.

Corpora vary along dimension

- Language: 7097 languages in the world
- Variety: like African American Language varieties.
 - AAE Twitter posts might include forms like "iont" (I don't)
- Code switching, e.g., Spanish/English, Hindi/English:

S/E: Por primera vez veo a @username actually being hateful! It was beautiful:
[For the first time I get to see @username actually being hateful! it was beautiful:]

H/E: dost tha or ra- hega ... dont wory ... but dherya rakhe
["he was and will remain a friend ... don't worry ... but have faith"]
- Genre: newswire, fiction, scientific articles, Wikipedia
- Author Demographics: writer's age, gender, ethnicity

Corpus datasheets

Motivation:

- (1) Why was the corpus collected?
- (2) By whom?
- (3) Who funded it?

Situation: In what situation was the text written?

Collection process: If it is a subsample how was it sampled? Was there consent? Pre-processing

Annotation process, language variety, demographics, etc.

Datasets vs. Corpora vs. Text Collections

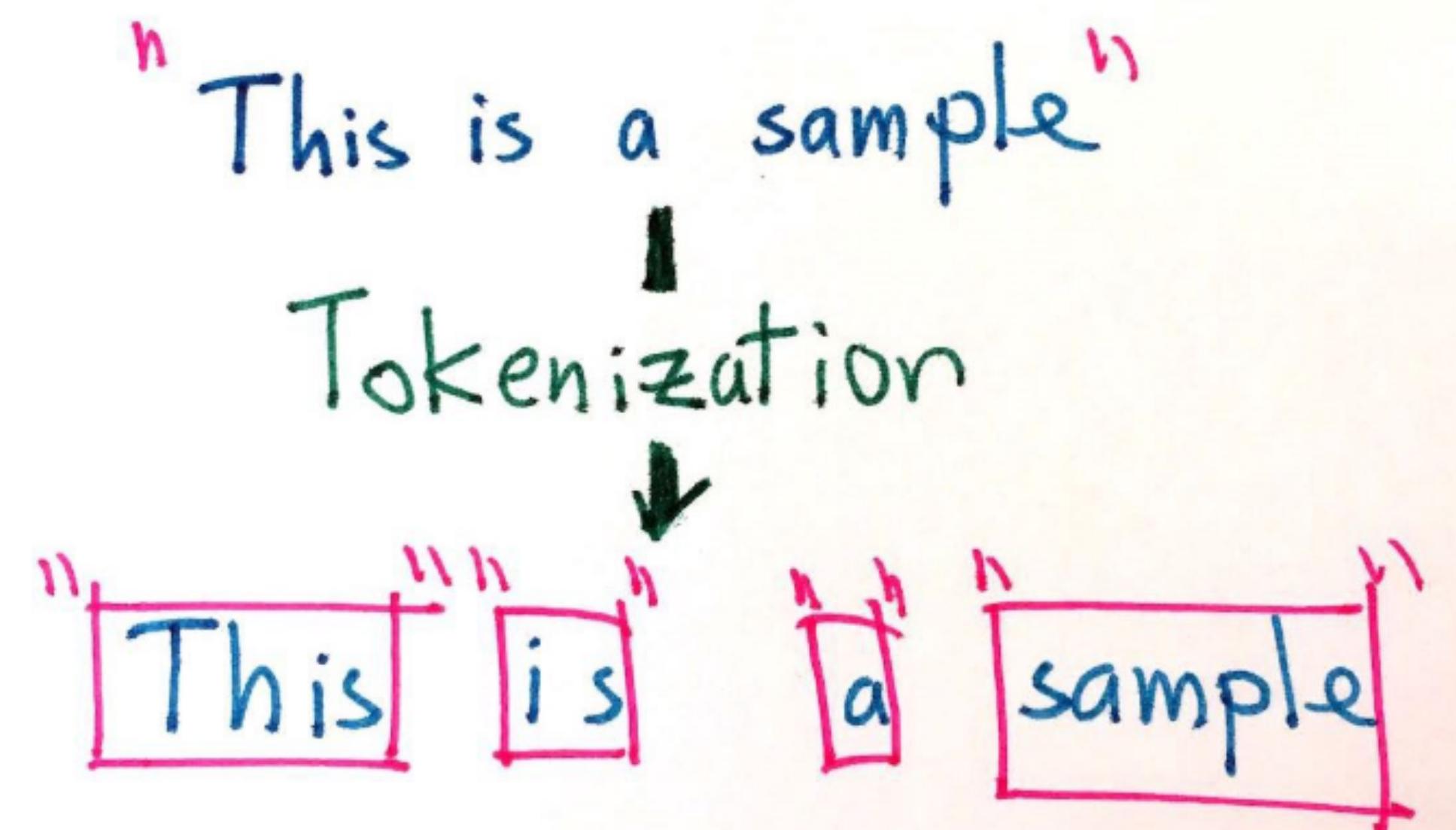
(1) Discussion: What is the difference?

Tokenization

Tokenization

recap

- (1) Breaking the input text into meaningful elements, smaller pieces or "tokens".
- (2) Tokens can be:
 - (1) elements for further semantic processing
 - (2) word, sentence, paragraph
- (3) Certain characters, such as punctuation marks, can be removed at the same time.



Regular Expressions: Recap

Regular expressions

(1) A formal language for specifying text strings

(2) How can we search for any of these?

(1) woodchuck

(2) woodchucks

(3) Woodchuck

(4) Woodchucks



Regular Expressions: Disjunctions

(1) Letters inside square brackets []

Pattern	Matches
[wW]oodchuck	Woodchuck, woodchuck
[1234567890]	Any digit

(3) Ranges [A-Z]

Pattern	Matches	
[A-Z]	An upper case	Drenched Blossoms
[a-z]	A lower case letter	my beans were impatient
[0-9]	A single digit	Chapter 1: Down the Rabbit Hole

Regular Expressions: Negation in Disjunction

(1) Negations [^Ss]

(1) Carat means negation only when first in []

Pattern	Matches	
[^A-Z]	Not an upper case letter	oyfn priпetchik
[^Ss]	Neither 'S' nor 's'	I have no exquisite reason"
[^e^]	Neither e nor ^	Look here
a^b	The pattern a carat	Look up <u>a^b</u> now

Regular Expressions: More Disjunction

- (1) Woodchuck is another name for groundhog!
- (2) The pipe | for disjunction

Pattern	Matches
groundhog woodchuck	woodchuck
yours mine	yours
a b c	= [abc]
[gG]roundhog [Ww]oodchuck	Woodchuck



Regular Expressions: ? *+ .

Pattern	Matches	
colou?r	Optional previous char	<u>color</u> <u>colour</u>
oo*h!	0 or more of previous char	<u>oh!</u> <u>ooh!</u> <u>oooh!</u> <u>ooooh!</u>
o+h!	1 or more of previous char	<u>oh!</u> <u>ooh!</u> <u>oooh!</u> <u>ooooh!</u>
baa+		<u>baa</u> <u>baaa</u> <u>baaaa</u>
beg.n		<u>begin</u> <u>begun</u> <u>begun</u>

Regular Expressions: Anchors ^ \$

Pattern	Matches
$^ [A-Z]$	Palo Alto
$^ [^A-Za-z]$	<u>1</u> "Hello"
\. \$	The end <u>.</u>
. \$	The end <u>?</u> The end <u>!</u>

Example

Find me all instances of the word “the” in a text

the

Misses capitalized examples

[tT]he

Incorrectly returns other or theology

[^a-zA-Z] [tT]he [^a-zA-Z]

Errors

The process we just went through was based on fixing two kinds of errors:

1. Matching strings that we should not have matched (**there, then, other**)
False positives (Type I errors)
2. Not matching things that we should have matched (**The**)
False negatives (Type II errors)

Errors cont.

- (1) In NLP we are always dealing with these kinds of errors.
- (2) Reducing the error rate for an application often involves two antagonistic efforts:
 - (1) Increasing accuracy or precision
(minimizing false positives)
 - (2) Increasing coverage or recall
(minimizing false negatives).

Summary

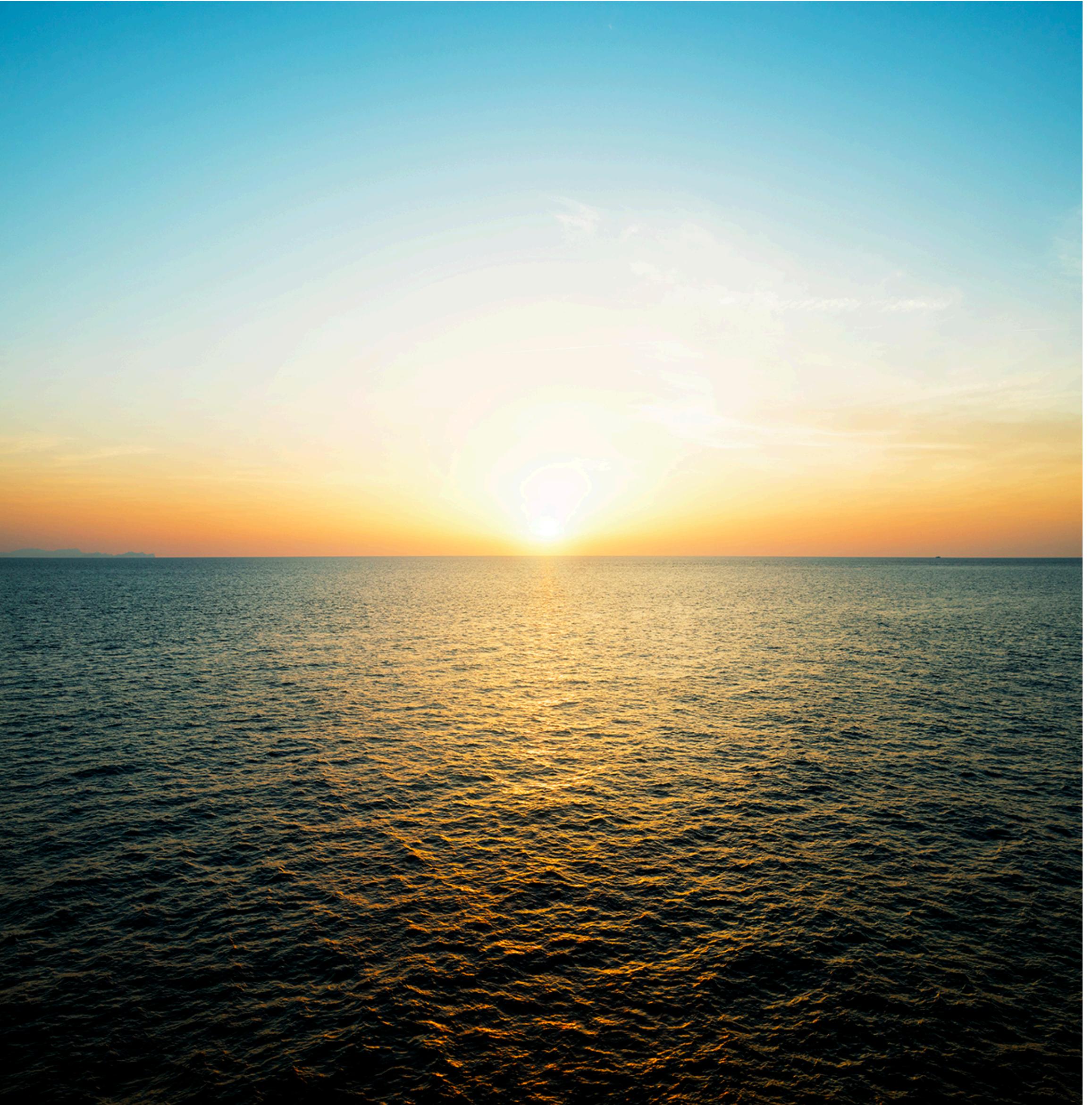
Regular expressions play a surprisingly large role

Sophisticated sequences of regular expressions are often the first model for any text processing text

For hard tasks, we use machine learning classifiers

But regular expressions are still used for pre-processing, or as features in the classifiers

Can be very useful in capturing generalizations



Break, 1 week

Text Normalization

Every NLP task requires text normalization:

1. Tokenizing (segmenting) words
2. Normalizing word formats
3. Segmenting sentences

Space-based tokenization

- (1) A very simple way to tokenize
 - (1) For languages that use space characters between words
 - (2) Segment off a token between instances of spaces
- (2) Unix tools for space-based tokenization
 - (1) The "tr" command
 - (2) Inspired by Ken Church's UNIX for Poets
 - (3) Given a text file, output the word tokens and their frequencies

Simple Tokenization in UNIX

(1)(Inspired by Ken Church's UNIX for Poets.)

(2)Given a text file, output the word tokens and their frequencies

```
tr -sc 'A-Za-z' '\n' < shakes.txt | sort | uniq -c
```

1945 A

72 AARON

19 ABBESS

5 ABBOT

... ...

25 Aaron

6 Abate

1 Abates

5 Abbess

6 Abbey

3 Abbot

.... ...

The first step: tokenizing

```
tr -sc 'A-Za-z' '\n' < shakes.txt | head
```

THE
SONNETS
by
William
Shakespeare
From
fairest
creatures
We
...

The second step: sorting

```
tr -sc 'A-Za-z' '\n' < shakes.txt | sort | head
```

A
A
A
A
A
A
A
A
A
A
...

More counting

(1) Merging upper and lower case

```
tr 'A-Z' 'a-z' < shakes.txt | tr -sc 'A-Za-z' '\n' | sort | uniq -c
```

(2) Sorting the counts

```
tr 'A-Z' 'a-z' < shakes.txt | tr -sc 'A-Za-z' '\n' | sort | uniq -c | sort -n -r
```

```
23243 the
```

```
22225 i
```

```
18618 and
```

```
16339 to
```

```
15687 of
```

```
12780 a
```

```
12163 you
```

```
10839 my
```

```
10005 in
```

```
8954 d
```

What happened here?

Issues in Tokenization

(1) Can't just blindly remove punctuation:

- (1) m.p.h., Ph.D., AT&T, cap'n
- (2) prices (\$45.55)
- (3) dates (01/02/06)
- (4) URLs (<http://www.stanford.edu>)
- (5) hashtags (#nlproc)
- (6) email addresses (someone@cs.colorado.edu)

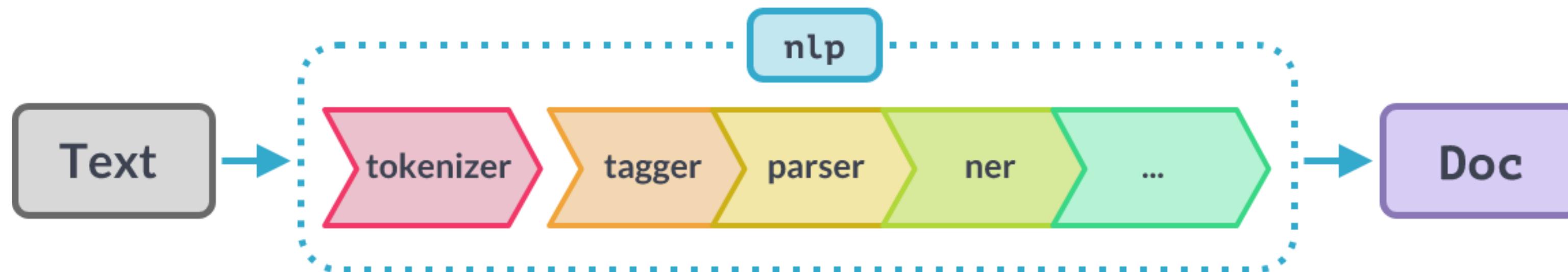
(2) Clitic: a word that doesn't stand on its own

- (1) "are" in we're, French "je" in j'ai, "le" in l'honneur

(3) When should multiword expressions (MWE) be words?

- (1) New York, rock 'n' roll

Tokenization in Spacy



```
import spacy  
nlp = spacy.load("en_core_web_md")  
doc = nlp("We are learning NLP using spaCy.")  
print ([token.text for token in doc])  
✓ 2.6s
```

```
['We', 'are', 'learning', 'NLP', 'using', 'spaCy', '.']
```

<https://medium.com/analytics-vidhya/nlp-with-spacy-tutorial-part-2-tokenization-and-sentence-segmentation-352df790a214>

<https://machinelearningknowledge.ai/complete-guide-to-spacy-tokenizer-with-examples/>

Tokenization in languages without spaces

Many languages (like Chinese, Japanese, Thai) don't use spaces to separate words!

How do we decide where the token boundaries should be?

Word tokenization in Chinese

Chinese words are composed of characters called "hanzi" (or sometimes just "zi")

Each one represents a meaning unit called a morpheme.

Each word has on average 2.4 of them.

But deciding what counts as a word is complex and not agreed upon.

李叶真的跟她的妈妈不一样，她看起来又跛又
廊，还经常生病。她总是喜欢生气，生气的时候总
是她。如果李叶的妈妈听到她生气，就会很生气。所
有的人都不喜欢这个孩子，他们从来没有让过这样
的孩子。为了不让李叶生气，她的阿姨总是很听李叶
的话。李叶说什么，她的阿姨就听她说什么。李叶
觉得在这个家里只有她的阿姨关心她。

sinosplice.com

How to do word tokenization in Chinese?

the table down there

姚明进入总决赛 “Yao Ming reaches the finals”

3 words?

姚明 进入 总决赛

YaoMing reaches finals

5 words?

姚 明 进 入 总 决 赛

Yao Ming reaches overall finals

7 characters? (don't use words at all):

姚 明 进 入 总 决 赛

Yao Ming enter enter overall decision game

Word tokenization / segmentation

In Chinese it's common to just treat each character (zi) as a token.

(1) So the **segmentation** step is very simple

In other languages (like Thai and Japanese), more complex word segmentation is required.

(2) The standard algorithms are neural sequence models trained by supervised machine learning.

Break, 5 min.

Byte Pair Encoding

Another option for text tokenization

Instead of

- (1)white-space segmentation
- (2)single-character segmentation

Use the data to tell us how to tokenize.

Subword tokenization (because tokens can be parts of words as well as whole words)

Subword tokenization

(1) Four common algorithms:

(1) **Byte-Pair Encoding (BPE)** (Sennrich et al., 2016)

(2) **Unigram language modeling tokenization** (Kudo, 2018)

(3) **WordPiece** (Schuster and Nakajima, 2012)

(4) **SentencePiece** (Kudo and Richardson, 2018) <https://arxiv.org/abs/1808.06226>

(2) All have 2 parts:

(1) A token **learner** that takes a raw training corpus and induces a vocabulary (a set of tokens).

(2) A token **segmenter** that takes a raw test sentence and tokenizes it according to that vocabulary

Byte Pair Encoding (BPE) token learner

Let vocabulary be the set of all individual characters

$$= \{A, B, C, D, \dots, a, b, c, d, \dots\}$$

- (1) Repeat:
 - (1) Choose the two symbols that are most frequently adjacent in the training corpus (say 'A', 'B')
 - (2) Add a new merged symbol 'AB' to the vocabulary
 - (3) Replace every adjacent 'A' 'B' in the corpus with 'AB'.
- (2) Until k merges have been done.

BPE token learner algorithm

- (1) We shall consider the following text:

low low low low low lowest
lowest newer newer newer
newer newer newer wider
wider wider new new

Algorithm 1 Learn BPE operations

```
import re, collections

def get_stats(vocab):
    pairs = collections.defaultdict(int)
    for word, freq in vocab.items():
        symbols = word.split()
        for i in range(len(symbols)-1):
            pairs[symbols[i],symbols[i+1]] += freq
    return pairs

def merge_vocab(pair, v_in):
    v_out = {}
    bigram = re.escape(' '.join(pair))
    p = re.compile(r'(?<!\\S)' + bigram + r'(?!\\S)')
    for word in v_in:
        w_out = p.sub(' '.join(pair), word)
        v_out[w_out] = v_in[word]
    return v_out

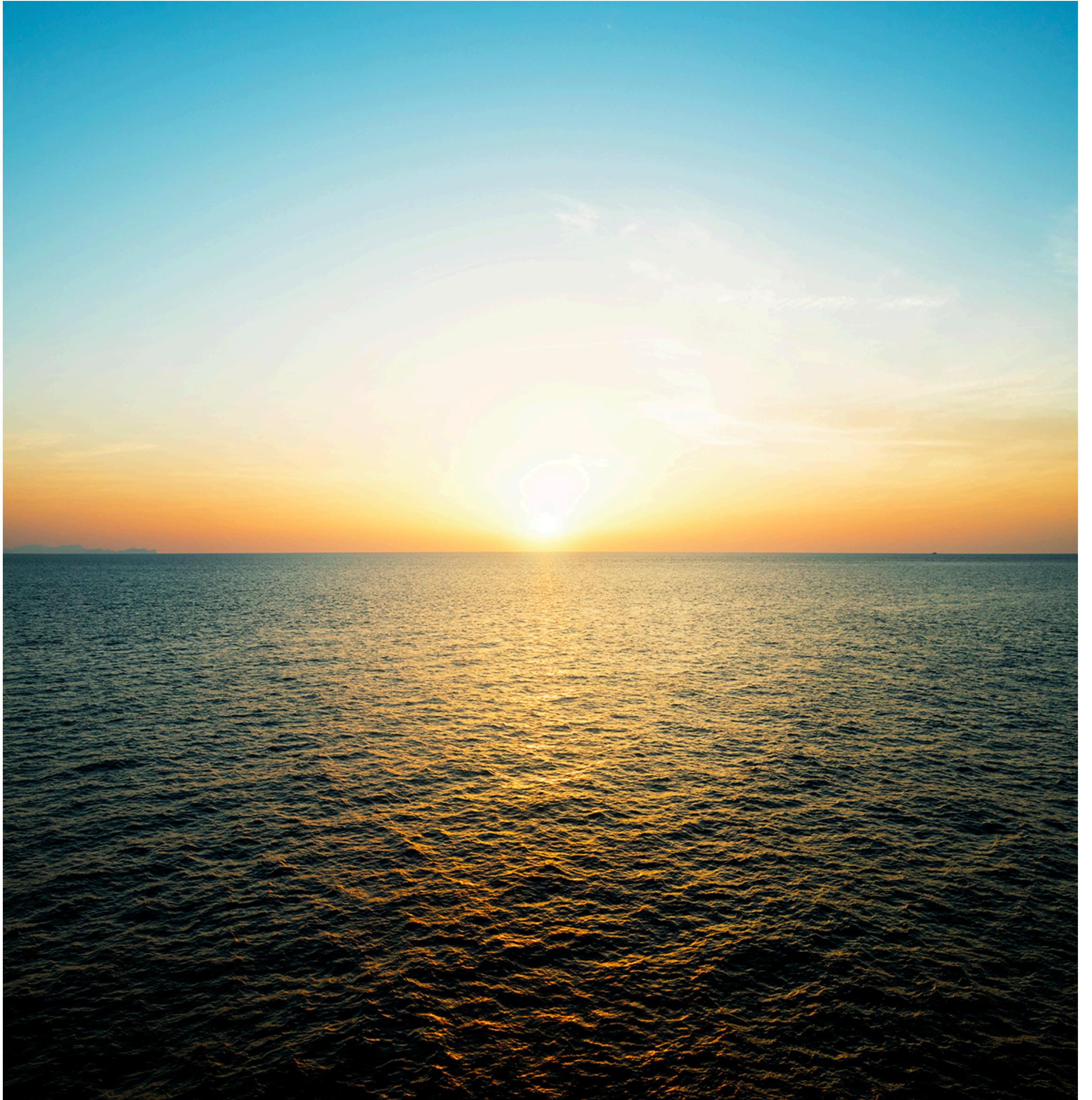
vocab = {'l o w </w>' : 5, 'l o w e r </w>' : 2,
         'n e w e s t </w>':6, 'w i d e s t </w>':3}
num_merges = 10
for i in range(num_merges):
    pairs = get_stats(vocab)
    best = max(pairs, key=pairs.get)
    vocab = merge_vocab(best, vocab)
    print(best)
```

Byte Pair Encoding (BPE)

Most subword algorithms are run inside space-separated tokens.

So we commonly first add a special end-of-word symbol '' before space in training corpus

Next, separate words into letters.



BPE token learner

Original (very fascinating 😳) corpus:

low low low low lowest lowest newer newer newer
newer newer newer wider wider wider new new

Add end-of-word tokens, resulting in this vocabulary:

corpus	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 _	

BPE: 1st Merge operation

corpus

5 low _
2 lowest _
6 newer _
3 wider _
2 new _

vocabulary

-, d, e, i, l, n, o, r, s, t, w

Merge e r to er

corpus

5 low _
2 lowest _
6 newer _
3 wider _
2 new _

vocabulary

-, d, e, i, l, n, o, r, s, t, w, er

BPE: 2nd Merge operation

corpus

5 low_
2 lowest_
6 newer_
3 wider_
2 new_

vocabulary

_, d, e, i, l, n, o, r, s, t, w, er

Merge er _ to er_

corpus

5 low_
2 lowest_
6 newer_
3 wider_
2 new_

vocabulary

, d, e, i, l, n, o, r, s, t, w, er, er

BPE: 3rd Merge operation

corpus

5 low_
2 lowest_
6 newer_
3 wider_
2 new_

vocabulary

, d, e, i, l, n, o, r, s, t, w, er, er

Merge n e to ne

corpus

5 low_
2 lowest_
6 newer_
3 wider_
2 new_

vocabulary

, d, e, i, l, n, o, r, s, t, w, er, er, ne

BPE

(1) The next merges are:

Merge	Current Vocabulary
(ne, w)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new
(l, o)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo
(lo, w)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo, low
(new, er_)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo, low, newer_
(low, _)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo, low, newer_, low_

BPE token segmenter algorithm

On the test data, run each merge learned from the training data:

- (1) Greedily
- (2) In the order we learned them
 - (test frequencies don't play a role)

So: merge every **e r** to **er**, then merge **er_** to **er_**, etc.

- (2) Result:
 - (1) Test set "n e w e r_" would be tokenized as a full word
 - (2) Test set "l o w e r_" would be two tokens: "low er_"

Properties of BPE tokens

Usually include frequent words

And frequent subwords

(1) Which are often morphemes like *-est* or *-er*

A morpheme is the smallest meaning-bearing unit of a language

(2) *unlikeliest* has 3 morphemes *un-*, *likely*, and *-est*

Word Normalization and other issues

Word Normalization

(1) Putting words/tokens in a standard format

(1) U.S.A. or USA

(2) uhhuh or uh-huh

(3) Fed or fed

(4) am, is, be, are

Case folding

- (1) Applications like IR: reduce all letters to lower case
 - (1) Since users tend to use lower case
 - (2) Possible exception: upper case in mid-sentence?
 - (1) e.g., ***General Motors***
 - (2) ***Fed*** vs. ***fed***
 - (3) ***SAIL*** vs. ***sail***
- (2) For sentiment analysis, MT, Information extraction
 - (1) Case is helpful (***US*** versus ***us*** is important)

Lemmatization

Represent all words as their lemma, their shared root

= dictionary headword form:

(1) *am, are, is* → *be*

(2) *car, cars, car's, cars'* → *car*

(3) Spanish **quiero** ('I want'), **quieres** ('you want')

→ **querer** 'want'

(1) *He is reading detective stories*

→ *He be read detective story*

Lemmatization is done by Morphological Parsing

(1)Morphemes:

- (1)The small meaningful units that make up words
- (2)**Stems**: The core meaning-bearing units
- (3)**Affixes**: Parts that adhere to stems, often with grammatical functions

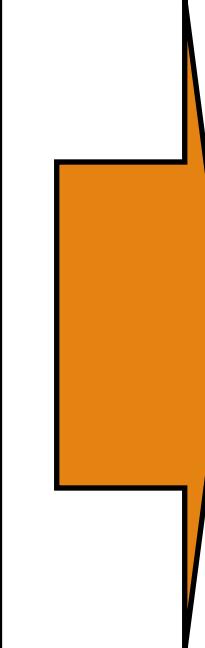
(2)Morphological Parsers:

- (1)Parse *cats* into two morphemes *cat* and *s*
- (2)Parse Spanish *amaren* ('if in the future they would love') into morpheme *amar* 'to love', and the morphological features *3PL* and *future subjunctive*.

Stemming

Reduce terms to stems, chopping off affixes crudely

This was not the map we found in Billy Bones's chest, but an accurate copy, complete in all things-names and heights and soundings-with the single exception of the red crosses and the written notes.



Thi wa not the map we found in Billi Bone s chest but an accur copi complet in all thing name and height and sound with the singl except of the red cross and the written note

.

Dealing with complex morphology is necessary for many languages

- (1) e.g., the Turkish word:
- (2) **Uygarlastiramadiklarimizdanmissinizcasina**
- (3) `(behaving) as if you are among those whom we could not civilize'
- (4) **Uygar** `civilized' + **las** `become'
 - + **tir** `cause' + **ama** `not able'
 - + **dik** `past' + **lar** 'plural'
 - + **imiz** 'p1pl' + **dan** 'abl'
 - + **mis** 'past' + **siniz** '2pl' + **casina** 'as if'

Sentence Segmentation

!, ? mostly unambiguous but period “.” is very ambiguous

- (1) Sentence boundary
- (2) Abbreviations like Inc. or Dr.
- (3) Numbers like .02% or 4.3

Common algorithm: Tokenize first: use rules or ML to classify a period as either (a) part of the word or (b) a sentence-boundary.

- (4) An abbreviation dictionary can help

Sentence segmentation can then often be done by rules based on this tokenization.

Summary