Natural Language Processing: Introduction and Preliminaries

CSE538 - Spring 2025

Instructor: H. Andrew Schwartz

1. Computers and Natural Language

2. Goal of NLP

3. Course Overview

4. Fundamentals Review
   a. Regular Expressions
   b. Probability Theory

5. Words and Corpora

uL8kLyze8kz.F8Yk(.eukuL8k?.zf!

```
t : u
h : L
e : 8
  : k
h : L
o : y
r : z
s : e
e : 8
  : k
r : z
a : .
c : F
e : 8
d : Y
  : k
p : (
a : .
s : e
t : u
  : k
t : u
h : L
e : 8
  : k
b : ?
a : .
r : z
n : f
. : !
```

uL8kLyze8kz.F8Yk(.eukuL8k?.zf!

the horse raced past the barn.

uL8kLyze8kz.F8Yk(.eukuL8k?.zf!

Most of modern NLP language understanding works by simply analyzing the patterns of language without any external knowledge. (over massive datasets and very large models)

uL8kLyze8kz.F8Yk(.eukuL8k?.zf!

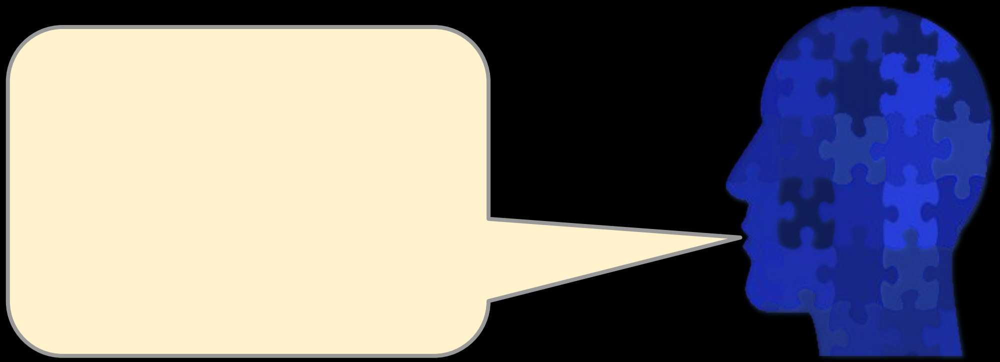
> 7 words? Likely a unique sequence

Google™

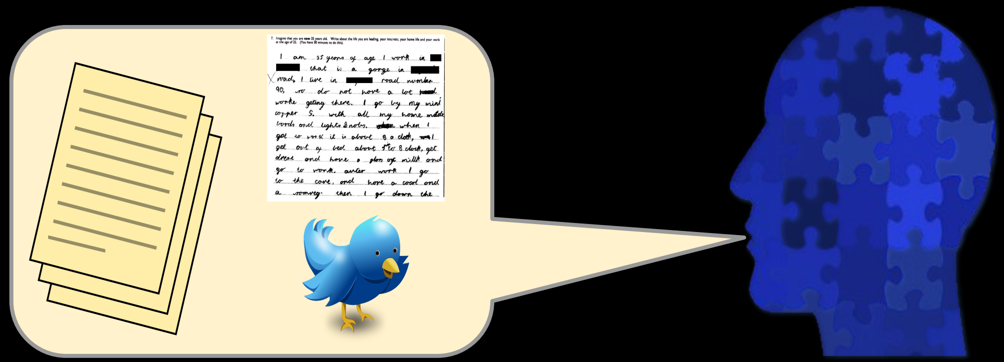
Most of modern NLP language understanding works by simply analyzing the patterns of language without any external knowledge. (over massive datasets and very large models)

Natural language is complicated!

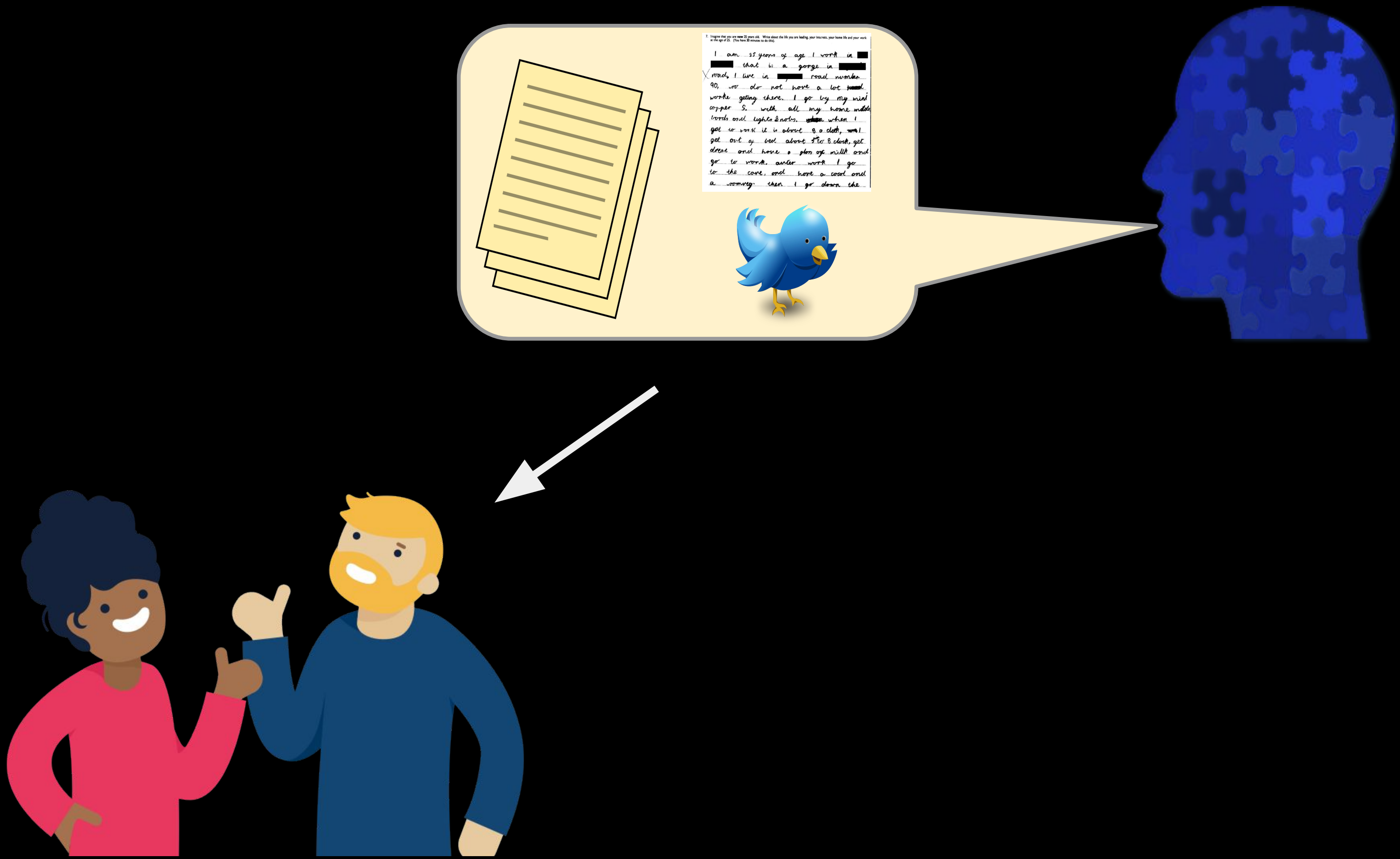
Natural language is complicated!



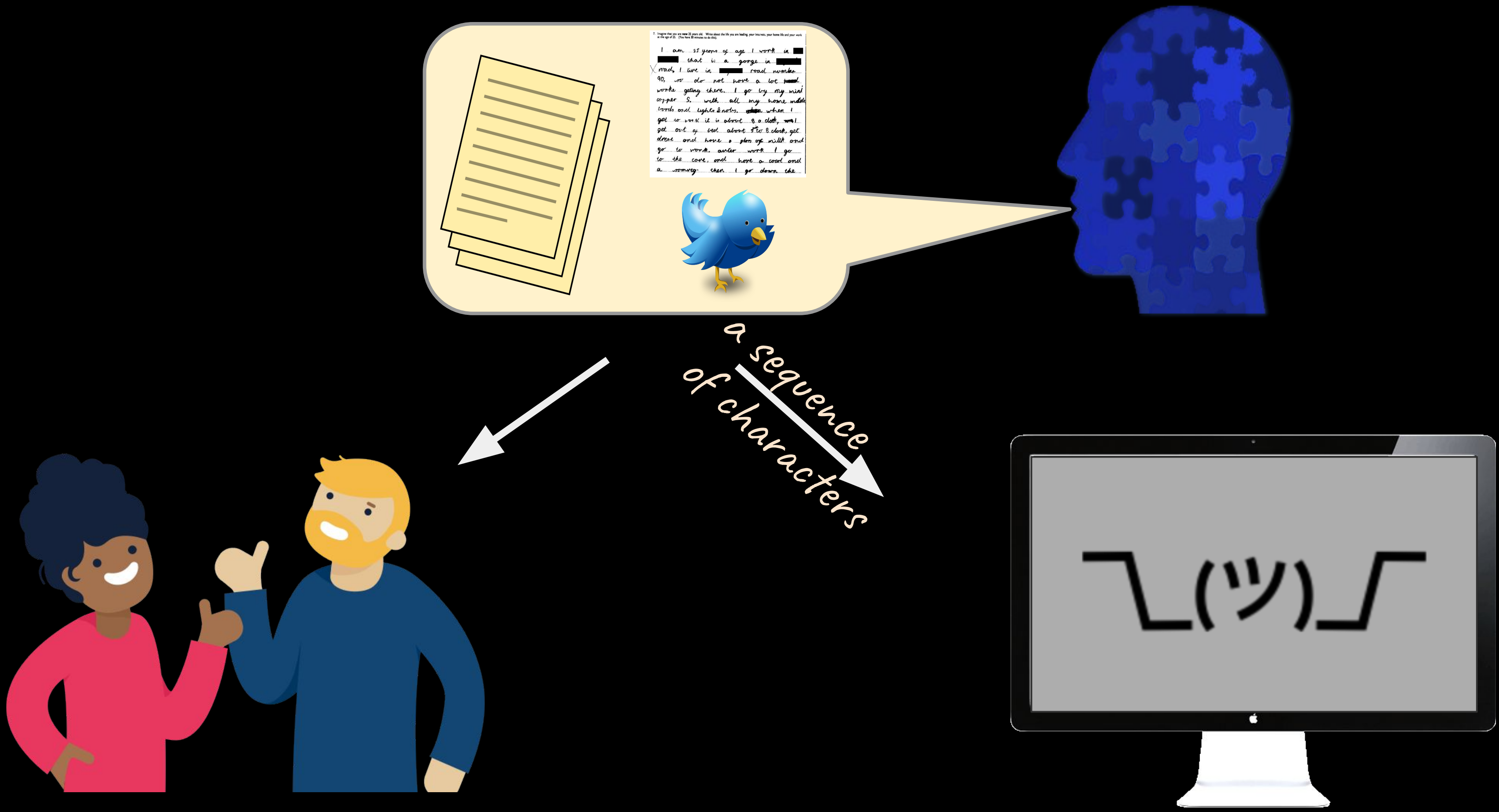
Natural language is complicated!



Natural language is complicated!



Natural language is complicated!



What is natural language like for a computer?

The horse raced past the barn.

What is natural language like for a computer?

The horse raced past the barn.

The horse raced past the barn fell.

What is natural language like for a computer?

The horse raced past the barn. ✓

The horse raced past the barn fell. ✓

What is natural language like for a computer?

The horse raced past the barn.

The horse raced past the barn fell.

The horse **runs** past the barn.

The horse **runs** past the barn fell.

What is natural language like for a computer?

The horse raced past the barn.

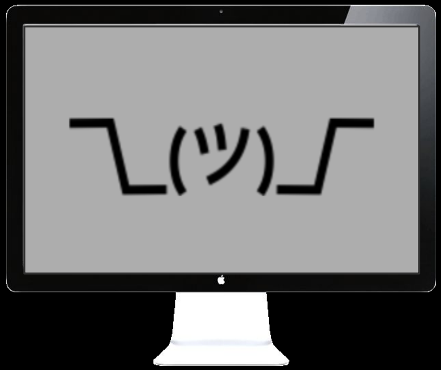
The horse **raced** past the barn fell.

that was

The horse **runs** past the barn.

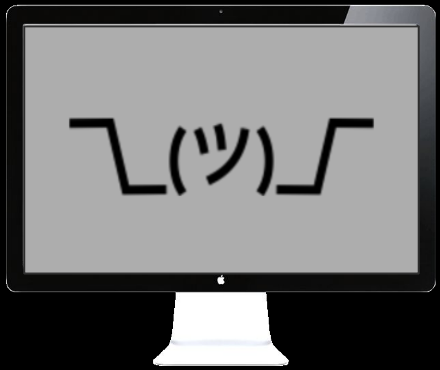
The horse **runs** past the barn fell.

More empathy for the computer...



Colorless purple ideas sleep furiously. (Chomsky, 1956; “purple”=> “green”)

More empathy for the computer...



Colorless purple ideas sleep furiously. (Chomsky, 1956; “purple”=> “green”)

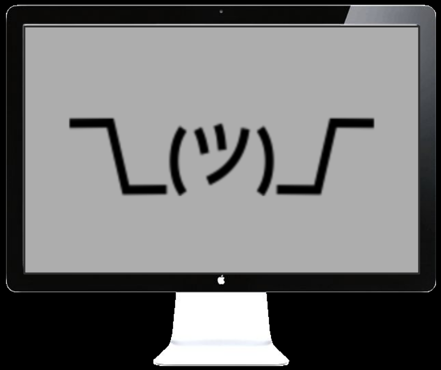
Fruit flies like a banana. Time flies like an arrow.

Daddy what did you bring that book that I don’t want to be read to out of up for?

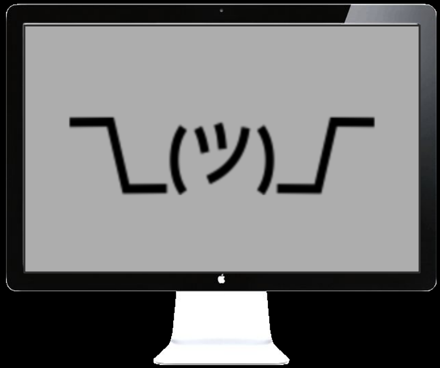
(Pinker, 1994)

More empathy for the computer...

She ate the cake with the frosting.



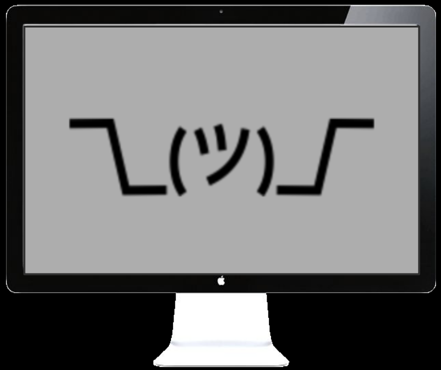
More empathy for the computer...



She ate the cake with the frosting.

[‘She’, ‘ate’, X, ‘with’, Y, ‘.’]

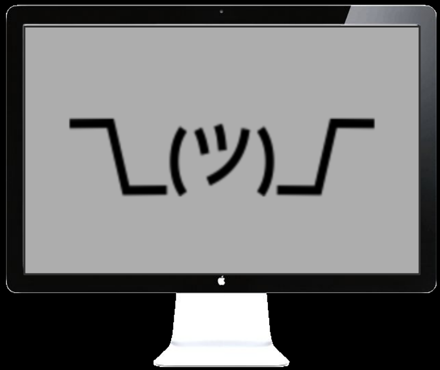
More empathy for the computer...



She ate the cake with the frosting.

[‘She’, ‘ate’, X, ‘with’, Y, ‘.’]
=> Y is a part of X

More empathy for the computer...

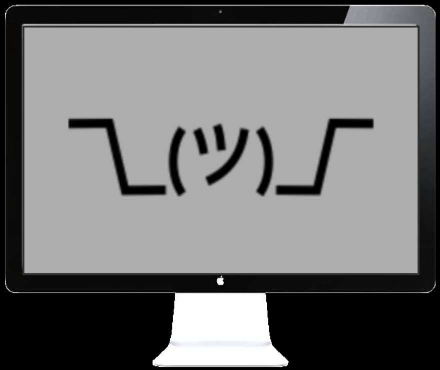


She ate the cake with the frosting.

She ate the cake with the fork.

```
['She', 'ate', X, 'with', Y, '.']
=> Y is a part of X
```

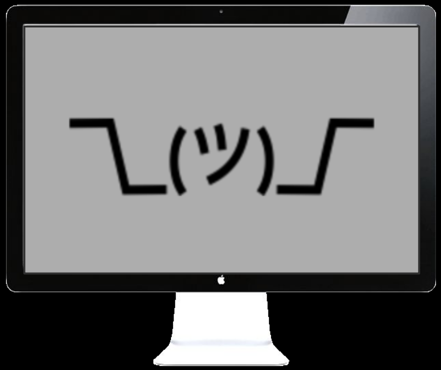
More empathy for the computer...



She ate the cake with the frosting.

She ate the cake with the fork.

More empathy for the computer...

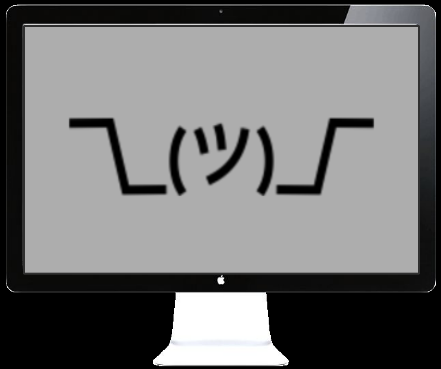


She ate the cake with the frosting.

She ate the cake with the fork.

He walked along the **port** next to the ship.

More empathy for the computer...



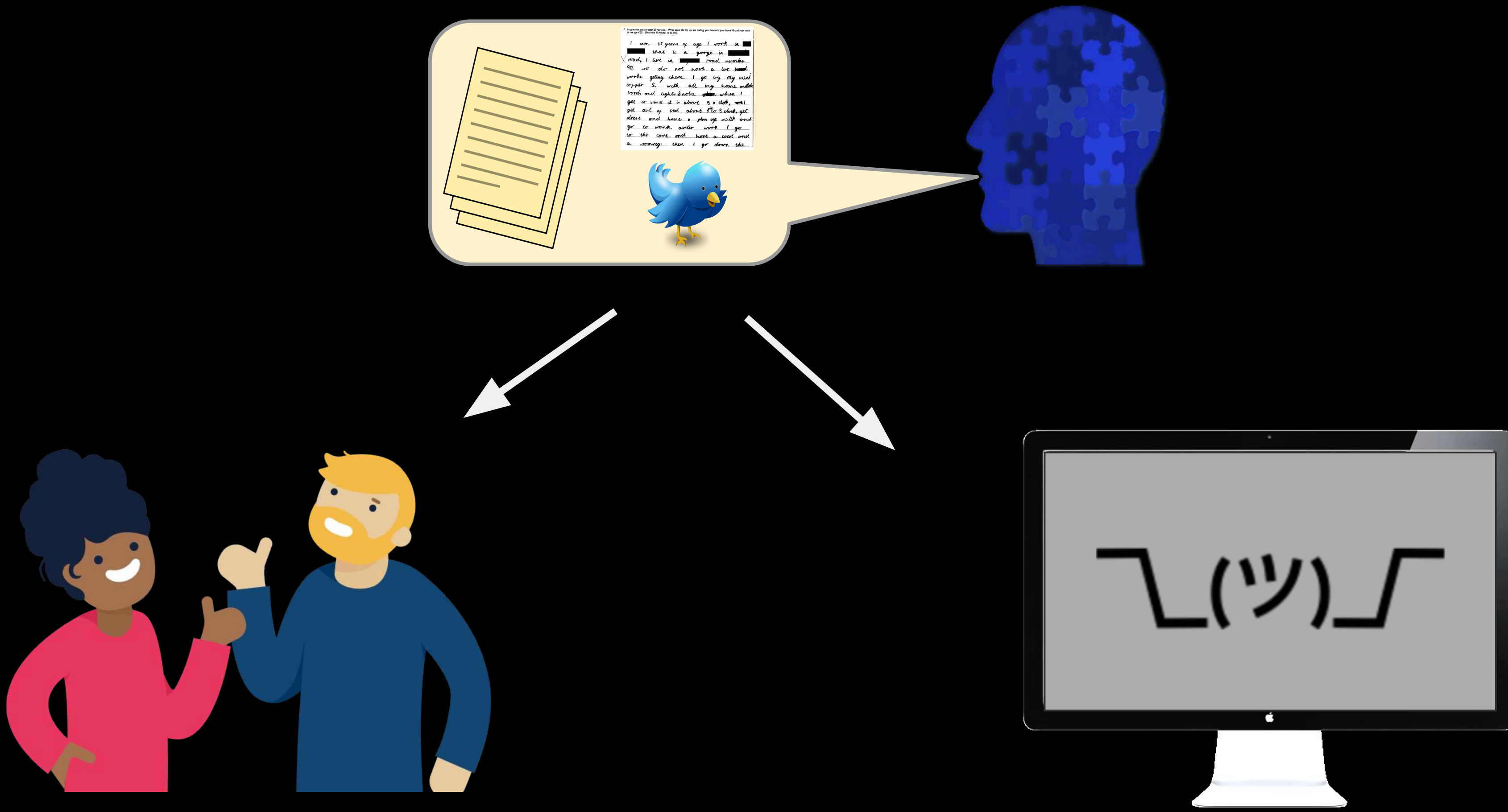
She ate the cake with the frosting.

She ate the cake with the fork.

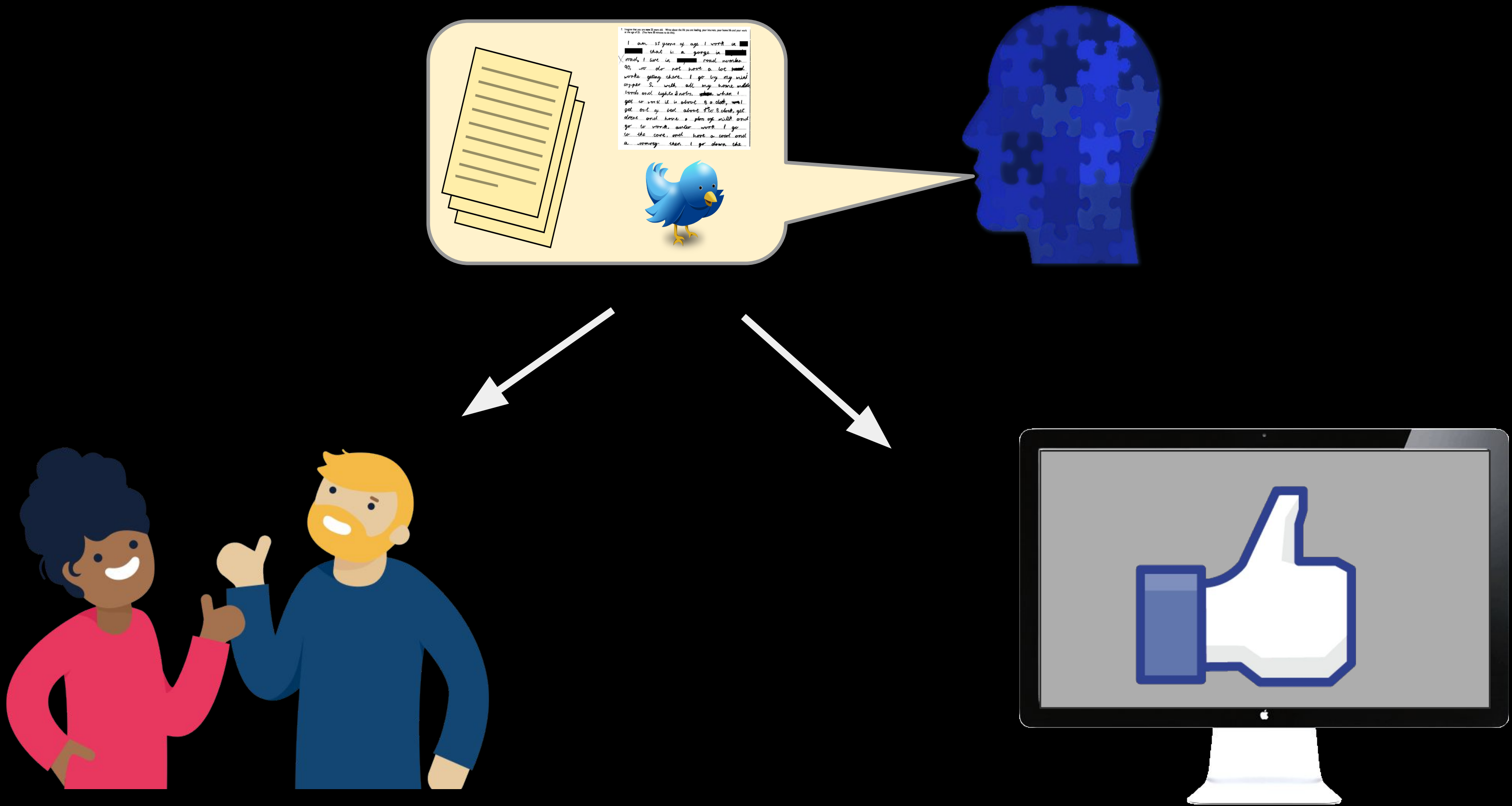
He put the **port** on the ship.

He walked along the **port** of the ship.

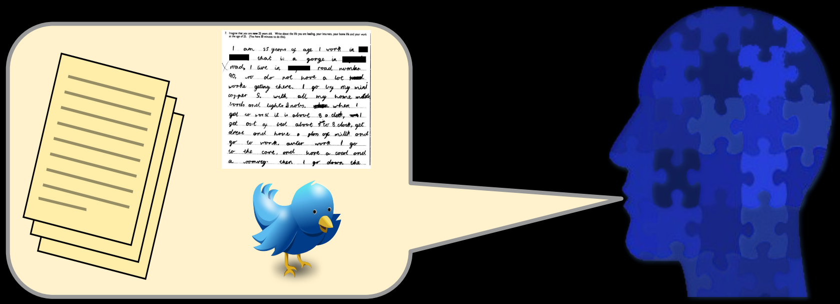
He walked along the **port** next to the ship.


¯_(ツ)_/¯

NLP's 0ld grand goal: completely understand natural language.

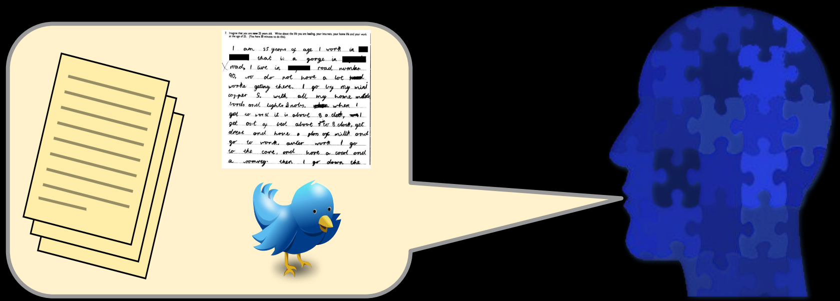


NLP’s practical applications <circa 2021>



- Machine translation

NLP's practical applications



- Machine translation

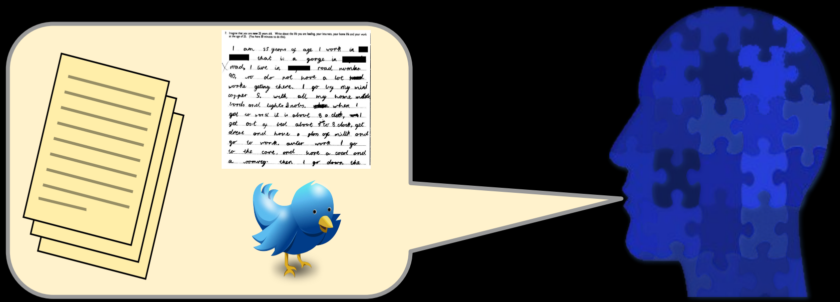
The spirit is willing, but the flesh is weak.

English -> Russian -> English

The vodka is good, but the meat is rotten.

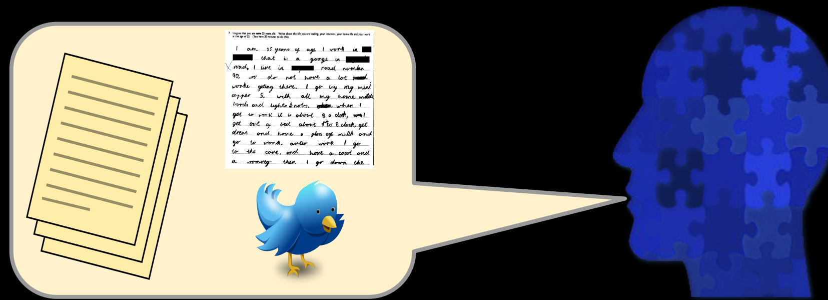
(Garbade, 2018)

NLP's practical applications



- Machine translation
- Sentiment Analysis

NLP's practical applications

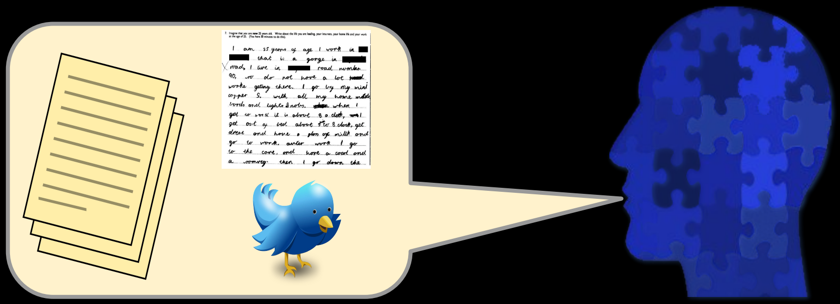


- Machine translation
- Sentiment Analysis

I like the the movie.

The movie is like terrible.

NLP’s practical applications

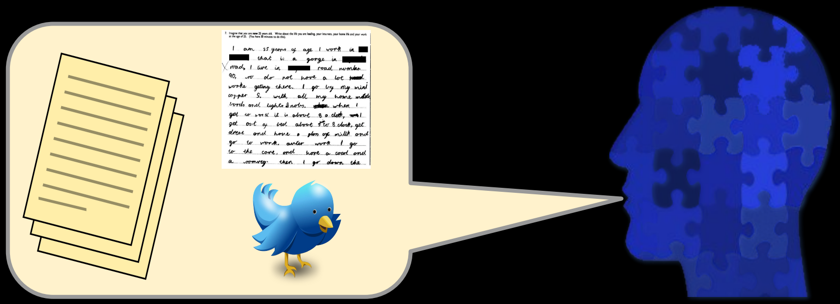


- Machine translation
- Sentiment Analysis
- Automatic speech recognition
    - Personalized assistants
    - Auto customer service

NLP's practical applications

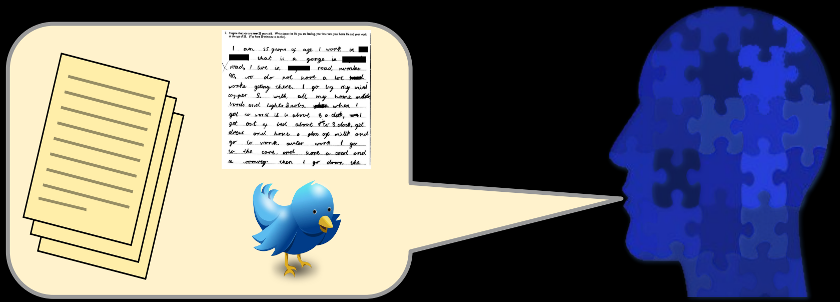
- Machine translation
- Sentiment Analysis
- Automatic speech recognition
    - Personalized assistants
    - Auto customer service

NLP's practical applications



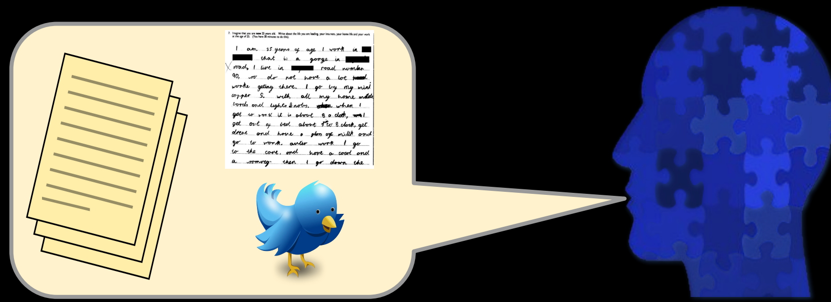
- Machine translation
- Sentiment Analysis
- Automatic speech recognition
  - Personalized assistants
  - Auto customer service
- Information Retrieval
  - Web Search
  - Question Answering

NLP's practical applications



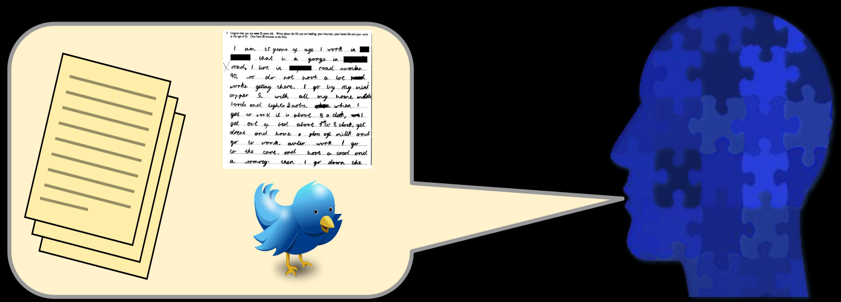
- Machine translation
- Sentiment Analysis
- Automatic speech recognition
  - Personalized assistants
  - Auto customer service
- Information Retrieval
  - Web Search
  - Question Answering

NLP's practical applications



- Machine translation
- Sentiment Analysis
- Automatic speech recognition
  - Personalized assistants
  - Auto customer service
- Information Retrieval
  - Web Search
  - Question Answering
- Computational Social Science

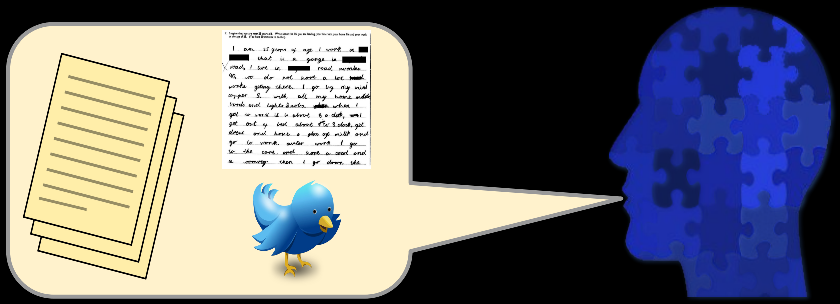
NLP’s practical applications



- Machine translation
- Sentiment Analysis
- Automatic speech recognition
  - Personalized assistants
  - Auto customer service
- Information Retrieval
  - Web Search
  - Question Answering
- Computational Social Science

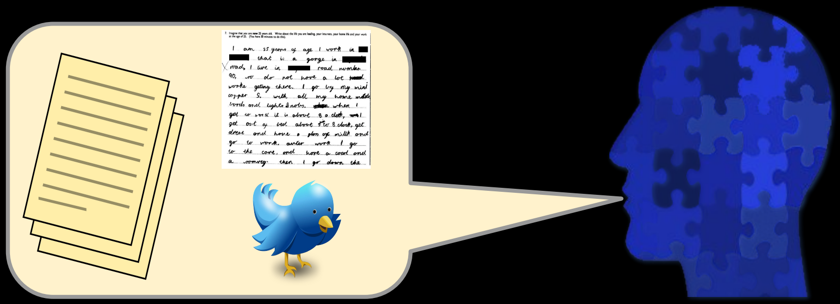
Schwartz, H. A., Eichstaedt, ... & Ungar. (2013). Personality, gender, and age in the language of social media: The open-vocabulary approach. *PloS one, 8(9)*.

NLP's practical applications



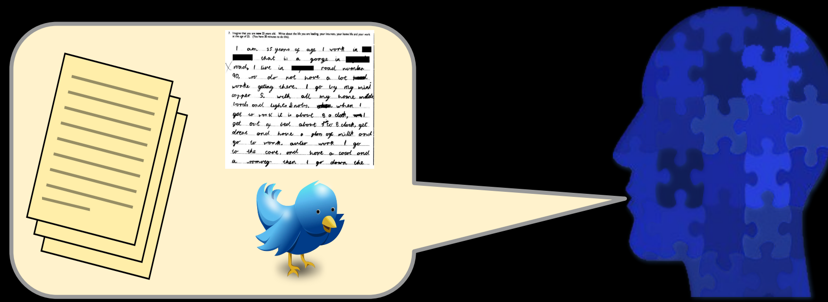
Schwartz, H. A., Eichstaedt, ... & Ungar. (2013). Personality, gender, and age in the language of social media: The open-vocabulary approach. *PloS one, 8(9)*.

NLP's practical applications



- Machine translation
- Sentiment Analysis
- Automatic speech recognition
  - Personalized assistants
  - Auto customer service
- Information Retrieval
  - Web Search
  - Question Answering
- Computational Social Science

NLP's practical applications

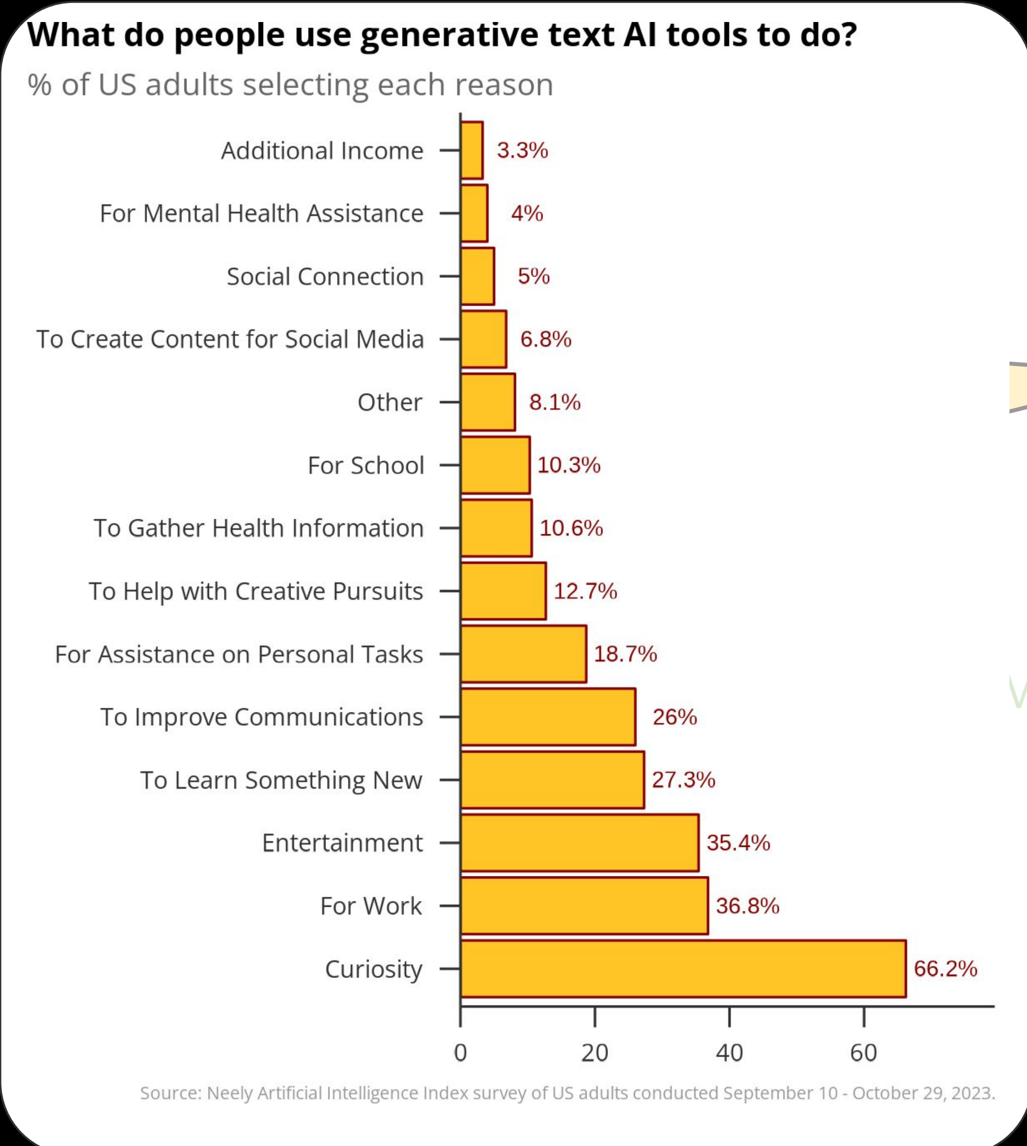


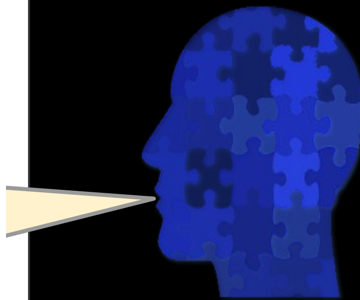
- Machine translation
- Sentiment Analysis
- Automatic speech recognition
  - Personalized assistants
  - Auto customer service
- Information Retrieval
  - Web Search
  - Question Answering
- Computational Social Science

LLMs have enabled:

- Open-ended information tasks. e.g.

  Editing emails
  Summarizing areas of work
  Question Answering
  Counseling (not well validated)
  …



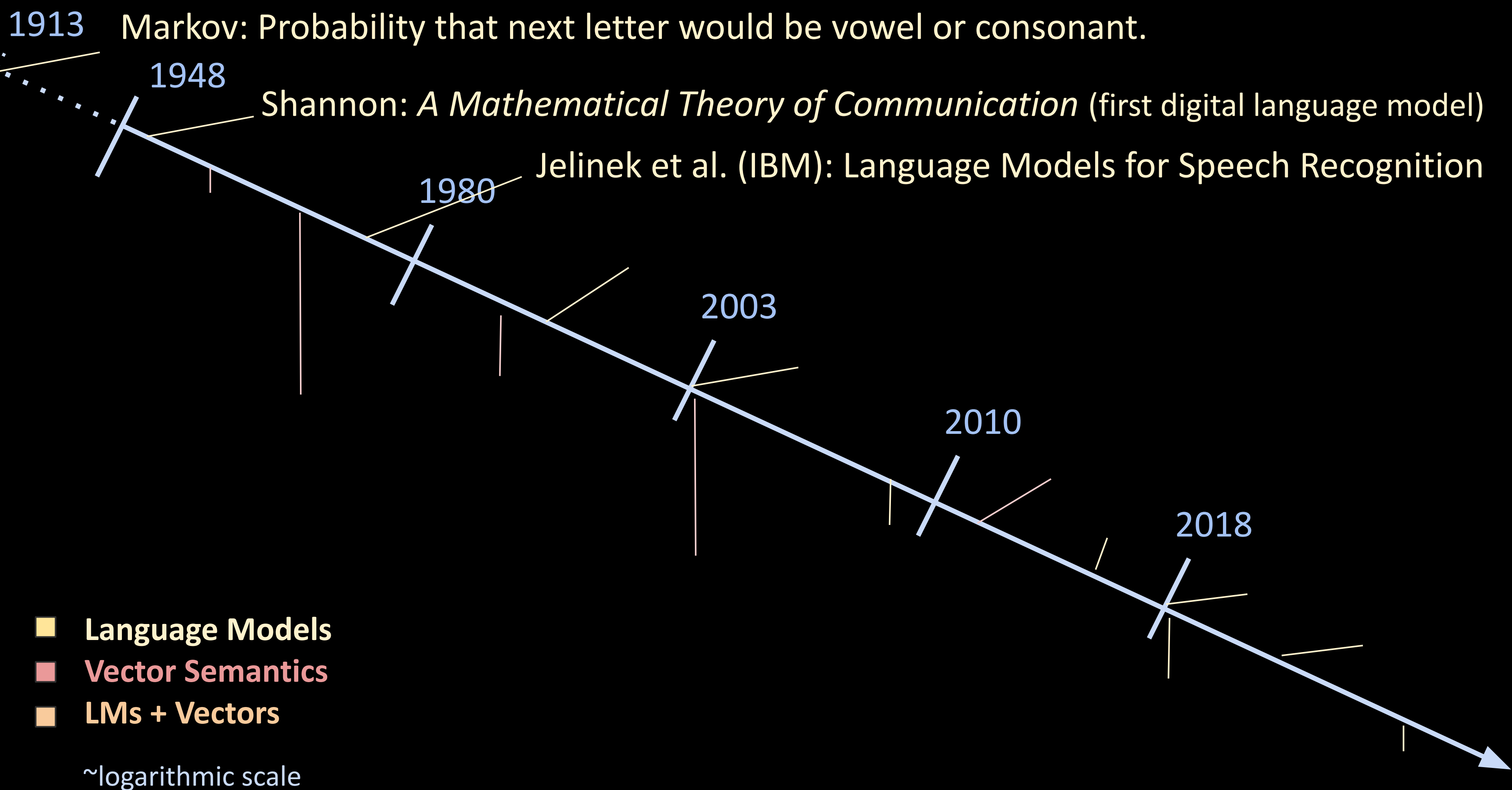


Ms have enabled:

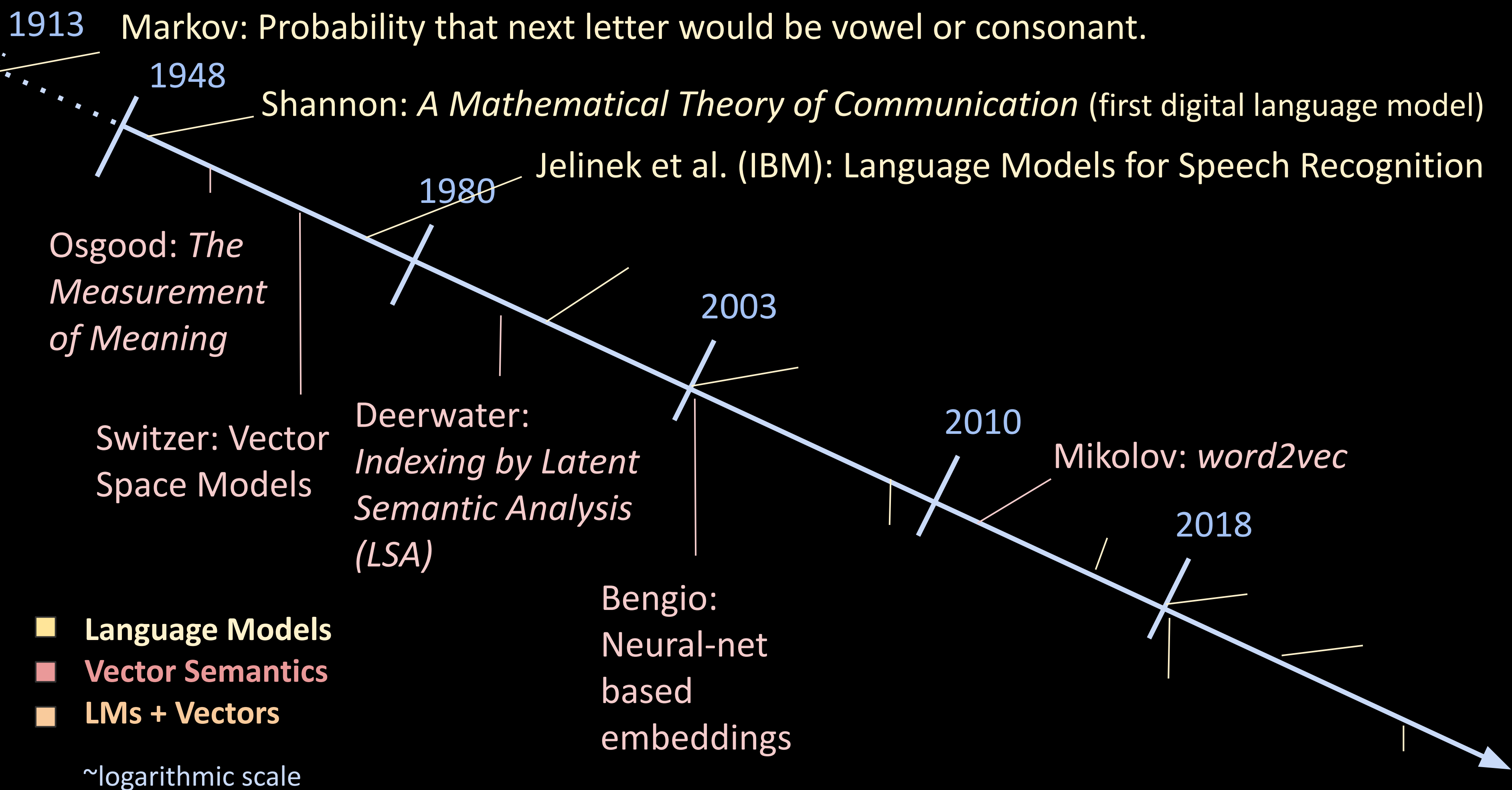
Open-ended information tasks. e.g.

Editing emails
Summarizing areas of work
Question Answering
Counseling (not well validated)
…

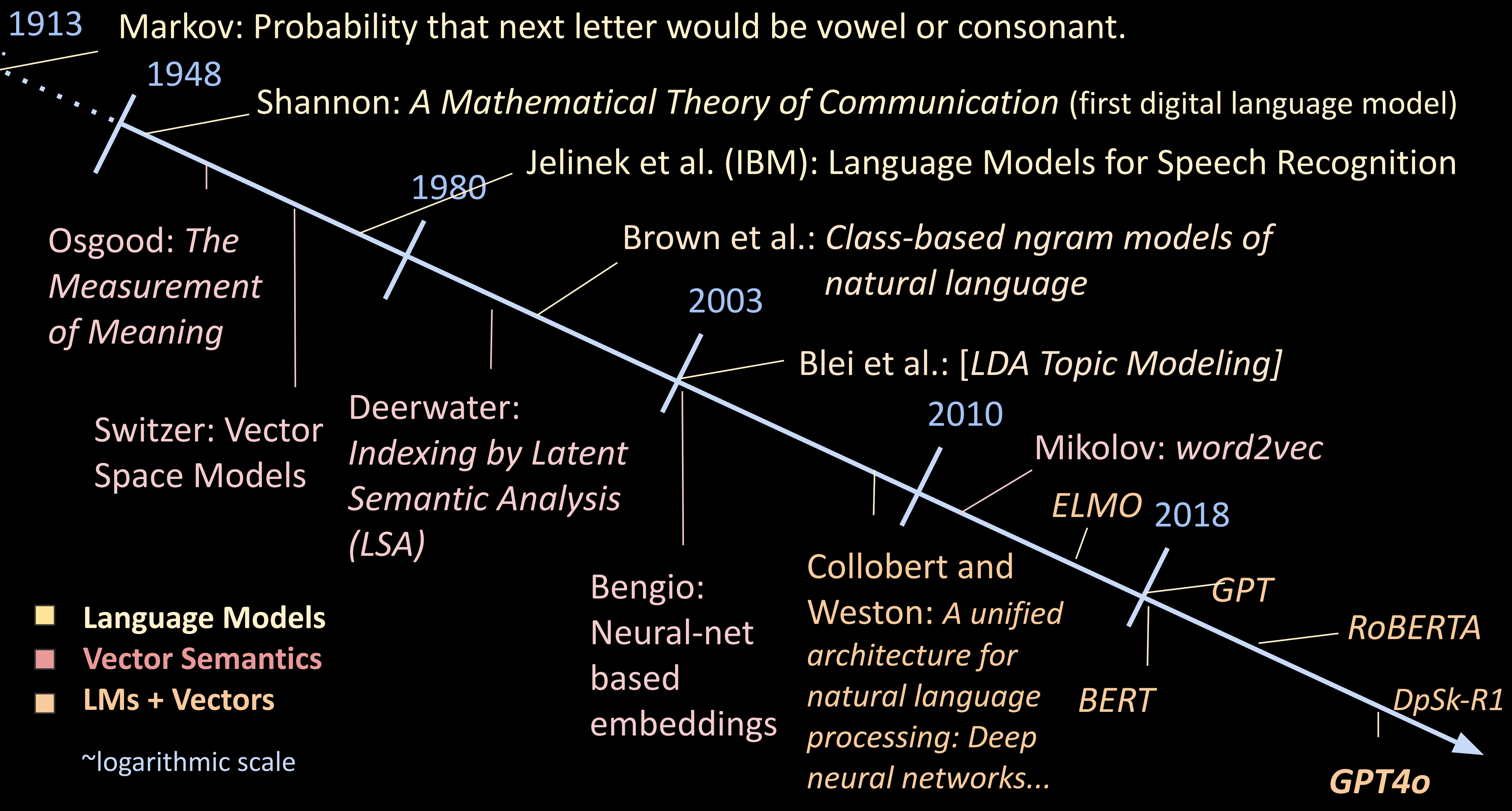
Timeline: *Language Modeling* and *Vector Semantics*



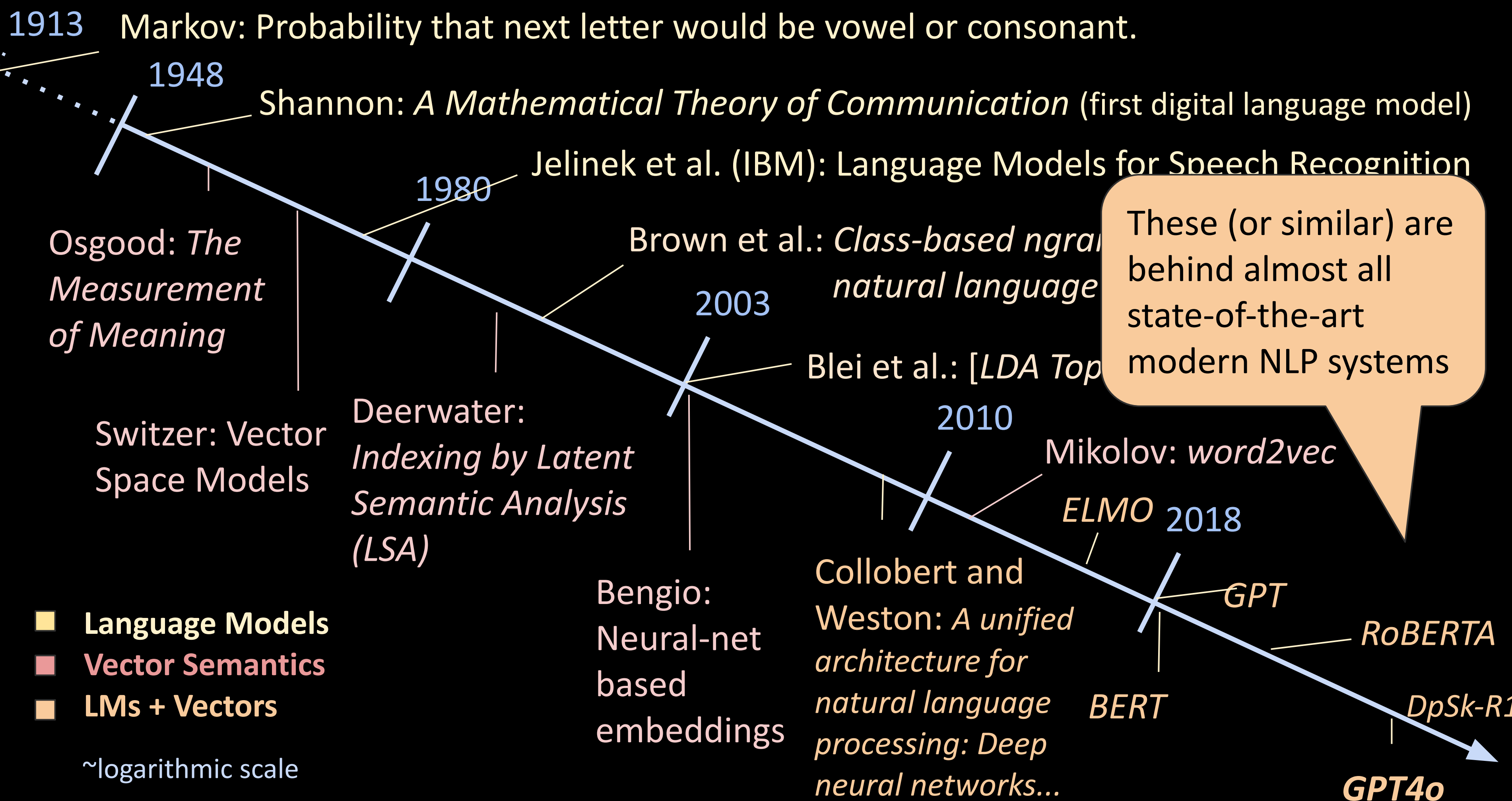
Timeline: *Language Modeling* and *Vector Semantics*



Timeline: *Language Modeling* and *Vector Semantics*



Timeline: *Language Modeling* and *Vector Semantics*



NLP: The Coarse

Speech and Language Processing

An Introduction to Natural Language Processing,
Computational Linguistics, and Speech Recognition
with Language Models

Third Edition draft

Daniel Jurafsky
Stanford University

James H. Martin
University of Colorado at Boulder

Draft of January 12, 2025. Comments and typos welcome!

web.stanford.edu/~jurafsky/slp3/

Course Website - Syllabus

www3.cs.stonybrook.edu/~has/CSE538/

Ingredients for success

The following covers the major components of the course and the estimated amount of time one might put into each if they are aiming to fully learn the material.

- ➔ **Review Quizzes:** 20 minutes, once a week (start second week)
- ➔ **Readings:** 2.5 hours/wk; 12 - 25 pages/wk (best before each class)
- ➔ **Study:** 1 - 2 hours/wk to review notes and look up extra content
- ➔ **Assignments (3):** 8 to 15 hours each
- ➔ **Get help early and be honest**: For anything you struggle to understand, seek office hours and extra learning suggestions.

Course Website - Syllabus

Typical grade distribution:

Grade	% of class
A	30%
A-	10%
B+	15%
B	15%
B-	10%
C+	10%
C	5%
C-	4%
F	1%

www3.cs.stonybrook.edu/~has/CSE538/

CSE538 - Preliminaries

Regular Expressions - a means for efficiently processing strings or sequences.

Use case: A basic tokenizer

Probability - a measurement of how likely an event is to occur.

Use case: How likely is “force” to be a noun?

Tokenizing Words:

tokens - an individual word instance.

types - distinct words.

CSE538 - Preliminaries

Regular Expressions - a means for efficiently processing strings or sequences.

  Use case: A basic tokenizer

Probability - a measurement of how likely an event is to occur.

  Use case: How likely is “force” to be a noun?

Tokenizing Words:

  tokens - an individual word instance.

  types - distinct words.

How many word tokens and word types?

Will, will Will will Will Will’s will?

Rose rose to put rose roes on her rows of roses.

Regular Expressions

The unsung hero of NLP

Regular Expressions

Patterns to match in a string.

Example:

pattern	**example strings**	**matches**
ing	‘kicking’, ‘ingles’, ‘class’	‘kick**ing**’, ‘**ing**les’, ‘class’X

Regular Expressions

Patterns to match in a string.

character class: [] --matches any single character inside brackets

pattern	**example strings**	**matches**
ing	‘kicking’, ‘ingles’, ‘class’	‘kick**ing**’, ‘**ing**les’, ‘class’X
[sS]bu	‘sbu’, ‘I like Sbu a lot’, ‘SBU’	

Regular Expressions

Patterns to match in a string.

character class: [] --matches any single character inside brackets

pattern	**example strings**	**matches**
ing	‘kicking’, ‘ingles’, ‘class’	‘kick**ing**’, ‘**ing**les’, ‘class’X
[sS]bu	‘sbu’, ‘I like Sbu a lot’, ‘SBU’	‘**sbu**’, ‘I like **Sbu** a lot’, ‘SBU’X

Regular Expressions

Patterns to match in a string.

character class: [] --matches any single character inside brackets

character ranges: [-] -- matches a range of characters according to ascii order

pattern	example strings	matches
ing	‘kicking’, ‘ingles’, ‘class’	‘kick**ing**’, ‘**ing**les’, ‘class’X
[sS]bu	‘sbu’, ‘I like Sbu a lot’, ‘SBU’	‘**sbu**’, ‘I like **Sbu** a lot’, ‘SBU’X
[A-Z][a-z]	‘sbu’, ‘Sbu’ #capital followed by lowercase	
[0-9][MmKk]	‘5m’, ‘50m’, ‘2k’, ‘2b’	

Regular Expressions

Patterns to match in a string.

character class: [] --matches any single character inside brackets

character ranges: [-] -- matches a range of characters according to ascii order

pattern	example strings	matches
ing	'kicking', 'ingles', 'class'	'kick**ing**', '**ing**les', 'class'X
[sS]bu	'sbu', 'I like Sbu a lot', 'SBU'	'**sbu**', 'I like **Sbu** a lot', 'SBU'X
[A-Z][a-z]	'sbu', 'Sbu' #capital followed by lowercase	'sbu'X, '**Sb**u'
[0-9][MmKk]	'5m', '50m', '2k', '2b'	'**5m**', '5**0m**', '**2k**', '2b'X

Regular Expressions

Patterns to match in a string.

character class: [] --matches any single character inside brackets
character ranges: [-] -- matches a range of characters according to ascii order
not characters: [^] -- matches any character except this

pattern	example strings	matches
ing	‘kicking’, ‘ingles’, ‘class’	‘kick**ing**’, ‘**ing**les’, ‘class’X
[sS]bu	‘sbu’, ‘I like Sbu a lot’, ‘SBU’	‘**sbu**’, ‘I like **Sbu** a lot’, ‘SBU’X
[A-Z][a-z]	‘sbu’, ‘Sbu’ #capital followed by lowercase	‘sbu’X, ‘**Sb**u’
[0-9][MmKk]	‘5m’, ‘50m’, ‘2k’, ‘2b’	‘**5m**’, ‘50m’X, ‘**2k**’, ‘2b’X
ing[^s]	‘kicking ’, ‘holdings ’, ‘ingles ’, ‘kicking’	

Regular Expressions

Patterns to match in a string.

character class: [] --matches any single character inside brackets
character ranges: [-] -- matches a range of characters according to ascii order
not characters: [^] -- matches any character except this

pattern	example strings	matches
ing	‘kicking’, ‘ingles’, ‘class’	‘kick**ing**’, ‘**ing**les’, ‘class’X
[sS]bu	‘sbu’, ‘I like Sbu a lot’, ‘SBU’	‘**sbu**’, ‘I like **Sbu** a lot’, ‘SBU’X
[A-Z][a-z]	‘sbu’, ‘Sbu’ #capital followed by lowercase	‘sbu’X, ‘**Sb**u’
[0-9][MmKk]	‘5m’, ‘50m’, ‘2k’, ‘2b’	‘**5m**’, ‘50m’X, ‘**2k**’, ‘2b’X
ing[^s]	‘kicking ’, ‘holdings ’, ‘ingles ’, ‘kicking’	‘kick**ing** ’, ‘holdings ’X, ‘**ingl**es’, ‘kicking’X

Regular Expressions

Patter

In python we denote regular expressions with:
r’PATTERN’

cha

character rang ... ording to ascii order

not characters ... atches any character except this

pattern	example strings	matches
r’ing’	‘kicking’, ‘ingles’, ‘class’	‘kick**ing**’, ‘**ing**les’, ‘class’X
r’[sS]bu’	‘sbu’, ‘I like Sbu a lot’, ‘SBU’	‘**sbu**’, ‘I like **Sbu** a lot’, ‘SBU’X
r’[A-Z][a-z]’	‘sbu’, ‘Sbu’ #capital followed by lowercase	‘sbu’X, ‘**Sb**u’
r’[0-9][MmKk]’	‘5m’, ‘50m’, ‘2k’, ‘2b’	‘**5m**’, ‘5**0m**’, ‘**2k**’, ‘2b’X
r’ing[^s]'	‘kicking ’, ‘holdings ’, ‘ingles ’	‘kick**ing** ’, ‘holdings ’X, ‘**ingl**es’

Regular Expressions

Matching recurring patterns:

* : match 0 or more

+ : match 1 or more

pattern	example strings	matches
r’ing!*’	‘swing’, ‘swing!’ ‘swing!!!’ ‘!!!’	
r’[sS][oO]+’	‘so’, ‘sooo’, ‘SOOoo’, ‘so!’, ‘soso’	

Regular Expressions

Matching recurring patterns:

* : match 0 or more

+ : match 1 or more

pattern	example strings	matches
r'ing!*'	'swing', 'swing!' 'swing!!!' '!!!'	'sw**ing**', 'sw**ing!**' 'sw**ing!!!**' '!!!'X
r'[sS][oO]+'	'so', 'sooo', 'SOOoo', 'so!', 'soso'	'**so**', '**sooo**', '**SOOoo**', '**so**!', '**so**''**so**' #would match twice

Regular Expressions

Matching recurring patterns:

* : match 0 or more
+ : match 1 or more
? : 0 or 1

pattern	example strings	matches
r'ing!*'	'swing', 'swing!' 'swing!!!' '!!!'	'sw**ing**', 'sw**ing!**' 'sw**ing!!!**' '!!!'X
r'[sS][oO]+'	'so', 'sooo', 'SOOoo', 'so!', 'soso'	'**so**', '**sooo**', '**SOOoo**', '**so**!', '**so**''**so**' #would match twice
r'oranges?'	'orange', 'oranges', 'orangess'	

Regular Expressions

Matching recurring patterns:

* : match 0 or more
+ : match 1 or more
? : 0 or 1

pattern	**example strings**	**matches**
r'ing!*'	'swing', 'swing!' 'swing!!!' '!!!'	'sw**ing**', 'sw**ing!**' 'sw**ing!!!**' '!!!'X
r'[sS][oO]+'	'so', 'sooo', 'SOOoo', 'so!', 'soso'	'**so**', '**sooo**', '**SOOoo**', '**so**!', '**so**''**so**' #would match twice
r'oranges?'	'orange', 'oranges', 'orangess'	'**orange**', '**oranges**', '**oranges**s' #matches all it can

Regular Expressions

Patterns applied to groups of characters

AA|BB : matches group AA or group BB

pattern	example strings	matches
r'hers|his|theirs"	'this is hers', 'this is his!'	'this is **hers**', 'this is **his**!'

Regular Expressions

Patterns applied to groups of characters

AA|BB : matches group AA or group BB
(AA) : apply any following operations to group

pattern	**example strings**	**matches**
r'hers|his'	'this is hers', 'this is his!'	'this is **hers**', 'this is **his**!'
r'([A-Z][a-z]+)+'	'This matches Cap Words followed By a Space.'	

Regular Expressions

Patterns applied to groups of characters

AA|BB : matches group AA or group BB
(AA) : apply any following operations to group

pattern	example strings	matches
r’hers|his’	‘this is hers’, ‘this is his!’	‘this is **hers**’, ‘this is **his**!’
r’([A-Z][a-z]+)+’	‘This matches Cap Words followed By a Space.’	‘**This** matches **Cap Words** followed **By** a Space.’

Regular Expressions

. : any single character

pattern	example strings	matches
.	‘kicking’	‘**k**’ ‘**i**’ ‘**c**’ ‘**k**’ ...

Regular Expressions

. : any single character

$: end of string

pattern	example strings	matches
.	‘kicking’	‘**k**’ ‘**i**’ ‘**c**’ ‘**k**’
.$	‘great’, ‘great!’, ‘50’	

Regular Expressions

. : any single character

$: end of string

pattern	example strings	matches
.	‘kicking’	‘**k**’ ‘**i**’ ‘**c**’ ‘**k**’
.$	‘great’, ‘great!’, ‘50’	‘grea**t**’, ‘great**!**’, ‘5**0**’

Regular Expressions

. : any single character
$: end of string
^: beginning of string

pattern	example strings	matches
.	‘kicking’	‘**k**’ ‘**i**’ ‘**c**’ ‘**k**’
.$	‘great’, ‘great!’, ‘50’	‘grea**t**’, ‘great**!**’, ‘5**0**’
^.a	‘Happy’, ‘slate’, ‘a’, ‘kick a door’	

Regular Expressions

. : any single character
$: end of string
^: beginning of string

pattern	example strings	matches
.	‘kicking’	‘**k**’ ‘**i**’ ‘**c**’ ‘**k**’
.$	‘great’, ‘great!’, ‘50’	‘grea**t**’, ‘great**!**’, ‘5**0**’
^.a	‘Happy’, ‘slate’, ‘a’, ‘kick a door’	‘**Ha**ppy’, ‘slate’, ‘a’X, ‘kick a door’
.a	‘Happy’, ‘slate’, ‘a’, ‘kick a door’	‘**Ha**ppy’, ‘s**la**te’, ‘a’X, ‘kick **a** door’

Regular Expressions

\s : matches any whitespace (space, tab, newline)
\b : matches a word boundary

Tokenizing -- breaking a sentence into simple lexical units (basically words).
Here are a couple simple regular expressions for tokenizing:

pattern	**example strings**	**matches**
r’(\s|^)[A-z]+...	‘Kick a door.’	

Regular Expressions

\s : matches any whitespace (space, tab, newline)
\b : matches a word boundary

Tokenizing -- breaking a sentence into simple lexical units (basically words).
Here are a couple simple regular expressions for tokenizing:

pattern	example strings	matches
r'(\s|^)[A-z]+([!\?\.]|$)?'	'Kick a door.'	

Regular Expressions

\s : matches any whitespace (space, tab, newline)
\b : matches a word boundary

Tokenizing -- breaking a sentence into simple lexical units (basically words).
Here are a couple simple regular expressions for tokenizing:

pattern	**example strings**	**matches**
r'(\s|^)[A-z]+([!\?\.]|$)?'	'Kick a door.'	'Kick' ' a' ' door.'

Regular Expressions

\s : matches any whitespace (space, tab, newline)
\b : matches a word boundary

Tokenizing -- breaking a sentence into simple lexical units (basically words).
Here are a couple simple regular expressions for tokenizing:

<table>
<tr><th>pattern</th><th>example strings</th><th>matches</th></tr>
<tr><td>r’(\s|^)[A-z]+([!\?\.]|$)?’</td><td>‘Kick a door.’</td><td>‘Kick’ ‘ a’ ‘ door.’</td></tr>
<tr><td>r’\b[A-z]+\b’</td><td>‘Kick a door.’</td><td>‘Kick', 'a', 'door'.’ #3 matches, no whitespace</td></tr>
</table>

Regular Expressions

```
import re
words = re.findall(r'\b[A-z]+\b', sentence)
for word in words:
    print(word)
```

pattern	example strings	matches
r’(\s|^)[A-z]+([!\?\.]|$)?’	‘Kick a door.’	‘**<u>Kick</u>**’ ‘**<u> a</u>**’ ‘**<u> door.</u>**’
r’\b[A-z]+\b’	‘Kick a door.’	‘**<u>Kick</u>** **<u>a</u>** **<u>door</u>**.’ #3 matches, no whitespace

Regular Expressions

```
import re

words = re.split(r'\s', sentence)

for word in words:

    print(word)
```

<table>
<tr><th>pattern</th><th>example strings</th><th>matches</th></tr>
<tr><td>r’(\s|^)[A-z]+([!\?\.]|$)?’</td><td>‘Kick a door.’</td><td>‘Kick’ ‘ a’ ‘ door.’</td></tr>
<tr><td>r’\b[A-z]+\b’</td><td>‘Kick a door.’</td><td>‘Kick a door.’ #3 matches, no whitespace</td></tr>
</table>

Probability

Probability

1970 — 1980s — 1990s — 2000s — 2010s — 2020s

Rule-based and Logic Systems (symbolic) — Statistical NLP — Machine Learning — Deep Learning (neural) — LLMs

Review: What is Probability?

Examples

1. outcome of flipping a coin
2. side of a die
3. mentioning a word
4. mentioning a word “a lot”

What is Probability?

What is Probability?

The chance that something will happen.

Given infinite observations of an event, the proportion of observations where a given outcome happens.

Strength of belief that something is true.

“Mathematical language for quantifying uncertainty” - Wasserman

What is Probability?

The chance that something will happen.

Given infinite observations of an event, the proportion of observations where a given outcome happens. *-- probability describes frequency in* ***data***

Strength of belief that something is true.

--probability describes amount of conviction toward a ***hypothesis***

“Mathematical language for quantifying uncertainty” - Wasserman

Probability

Ω : Sample Space, set of all outcomes of a random experiment

A : Event (***A*** $\subseteq$ **Ω**), collection of possible outcomes of an experiment

P(*A*): Probability of event ***A*, P** is a function: events$\to\mathbb{R}$

Probability

Ω : Sample Space, set of all outcomes of a random experiment

A : Event (***A*** ⊆ **Ω**), collection of possible outcomes of an experiment

P(*A*): Probability of event ***A*, P** is a function: events→ℝ

1. **P(Ω)** = 1
2. **P(*A*)** ≥ 0 **,** for all ***A***

If A_1, A_2, … are disjoint events then: $$\mathrm{P}(\bigcup_i^{\infty} A_i) = \sum_i^{\infty} \mathrm{P}(A_i)$$

Probability

Ω : Sample Space, set of all outcomes of a random experiment

A : Event (***A*** ⊆ **Ω**), collection of possible outcomes of an experiment

P(*A*): Probability of event ***A*, P** is a function: events→ℝ

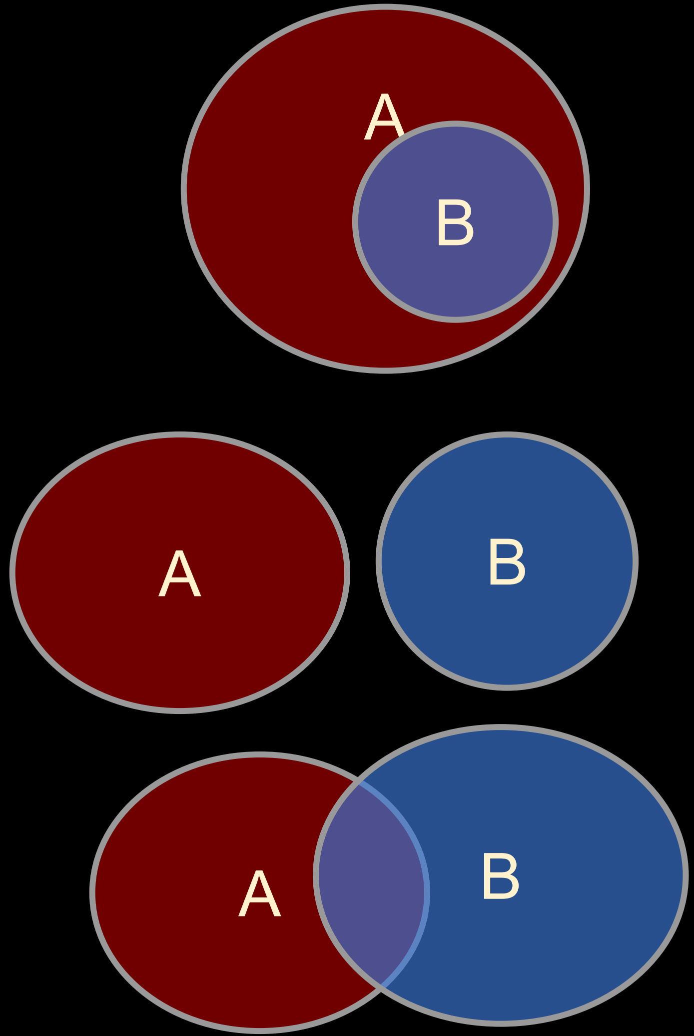
P is a *probability measure*, if and only if

1. **P(Ω)** = 1
2. **P(*A*)** ≥ 0 **,** for all ***A***

If A_1, A_2, … are disjoint events then: $$\mathrm{P}\left(\bigcup_{i}^{\infty} A_i\right) = \sum_{i}^{\infty} \mathrm{P}(A_i)$$

Probability

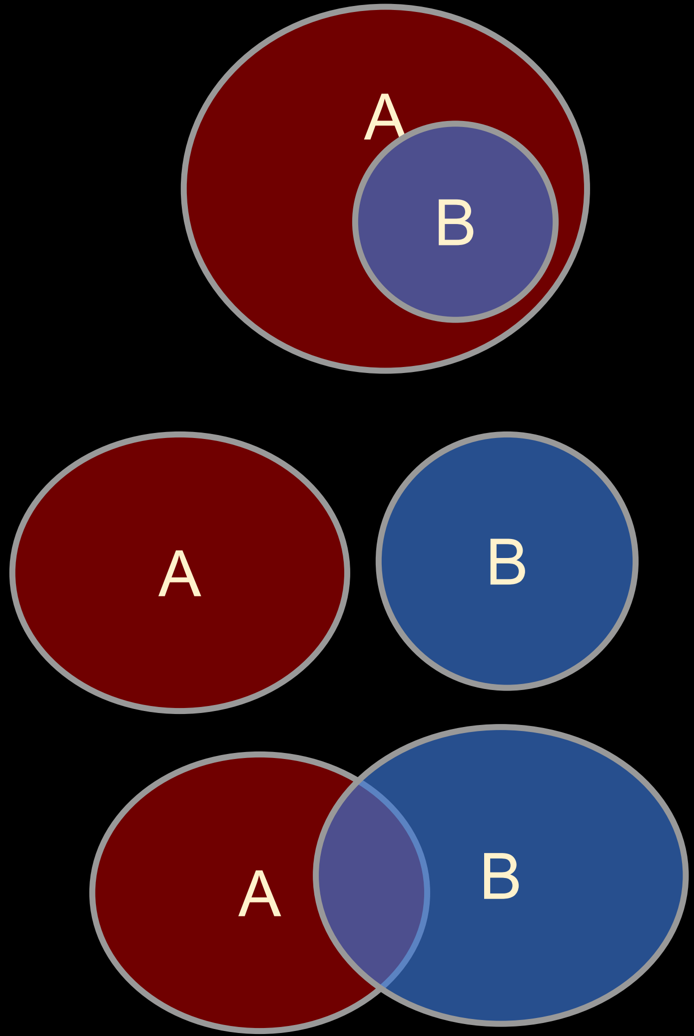
Some Properties:



Probability

Some Properties:

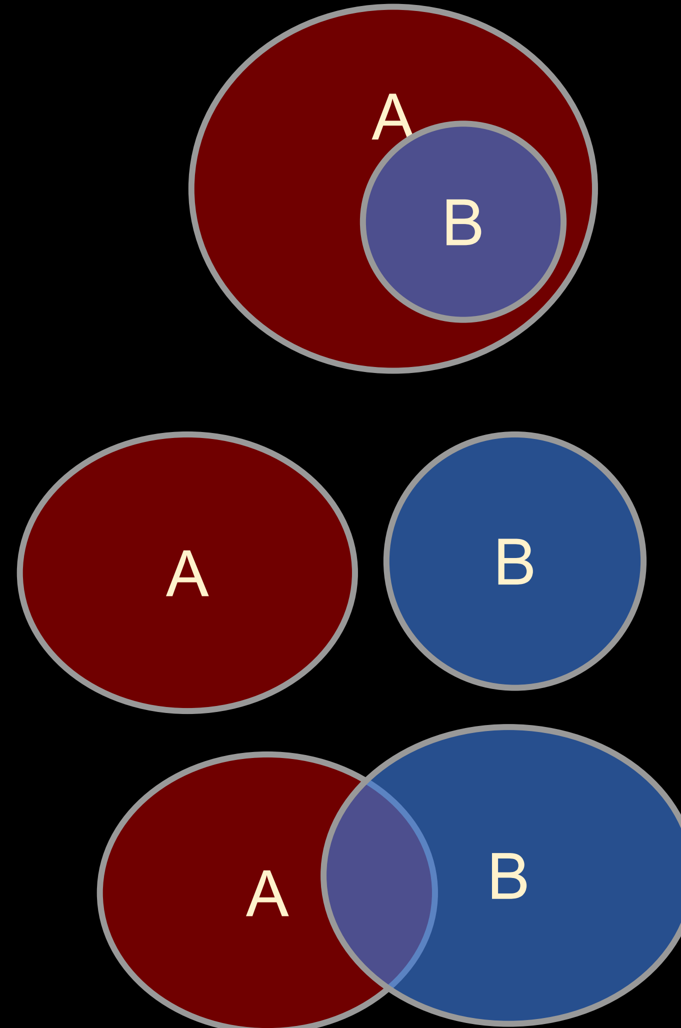
1. If $B \subseteq A$ then $\mathbf{P}(A) \geq \mathbf{P}(B)$



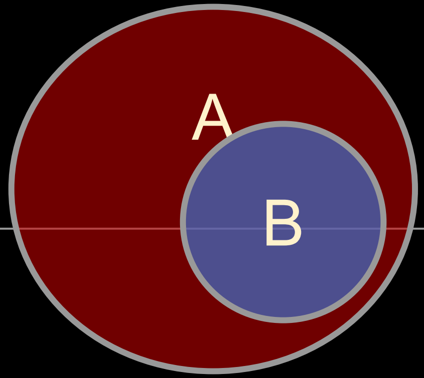
Probability

Some Properties:

1. If $B \subseteq A$ then $P(A) \geq P(B)$
2. $P(A \cup B) \leq P(A) + P(B)$



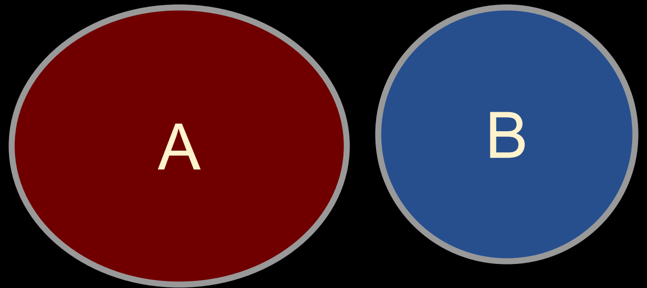
Probability

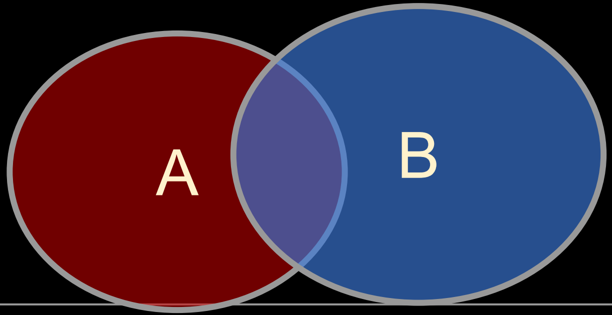


Some Properties:

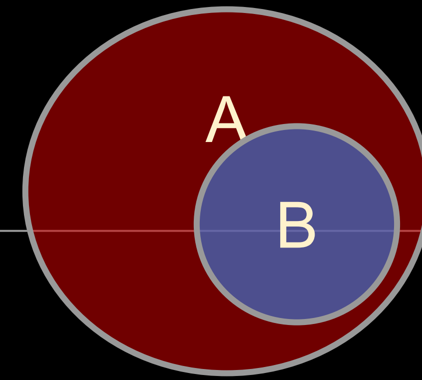
1. If $B \subseteq A$ then $P(A) \geq P(B)$
2. $P(A \cup B) \leq P(A) + P(B)$
3. $P(A \cap B) \leq \min(P(A), P(B))$
4. $P(\neg A) = P(\Omega / A) = 1 - P(A)$

/ is set difference





Probability

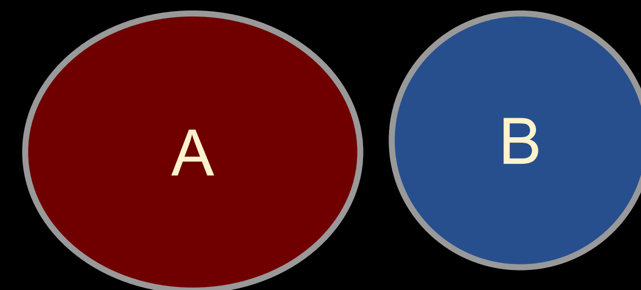


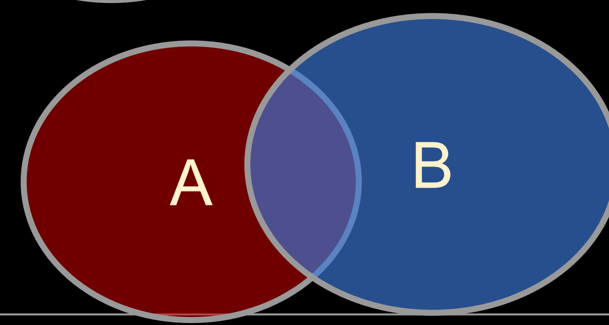
Some Properties:

1. If $B \subseteq A$ then $P(A) \geq P(B)$
2. $P(A \cup B) \leq P(A) + P(B)$
3. $P(A \cap B) \leq \min(P(A), P(B))$
4. $P(\neg A) = P(\Omega / A) = 1 - P(A)$

/ is set difference

$P(A \cap B)$ will be notated as $P(A, B)$





Probability

Independence

Two Events: A and B

Does knowing something about A tell us whether B happens (and vice versa)?

Probability

Independence

Two Events: *A* and *B*

Does knowing something about *A* tell us whether *B* happens (and vice versa)?

1. A: first flip of a fair coin; B: second flip of the same fair coin
2. A: sentence mentions (or not) the word “happy”
   B: sentence mentions (or not) the word “birthday”

Probability

Independence

Two Events: *A* and *B*

Does knowing something about *A* tell us whether *B* happens (and vice versa)?

1. A: first flip of a fair coin; B: second flip of the same fair coin
2. A: sentence mentions (or not) the word “happy”
   B: sentence mentions (or not) the word “birthday”

Two events, A and B, are ***independent*** iff: **P(*A*, *B*) = P(*A*)P(*B*)**

Probability

Conditional Probability

$$P(A|B) = \frac{P(A, B)}{P(B)}$$

Probability

Conditional Probability

$$P(A|B) = \frac{P(A, B)}{P(B)}$$

"|" is often referred to as "given":

"*The probability of A* ***given*** *B is ...*"

Probability

Conditional Probability

$$P(A|B) = \frac{P(A, B)}{P(B)}$$

Two events, A and B, are ***independent*** iff: $P(A, B) = P(A)P(B)$

$P(A, B) = P(A)P(B)$ iff $P(B|A) = P(B)$

Interpretation of Independence:

Observing *A* has no effect on probability of *B.*

(Disjoint events, typically, are not independent!)

Probability

Conditional Probability

$$P(A|B) = \frac{P(A, B)}{P(B)}$$

Independence example:

F1=H: first flip of a fair coin is heads
F2=H: second flip of the same coin is heads
P(F1=H) = 0.5 P(F2=H) = **0.5**
P(F2=H, F1=H) = 0.25

Two events, A and B, are ***independent*** iff: $\mathbf{P(A, B) = P(A)P(B)}$

$\mathbf{P(A, B) = P(A)P(B)}$ iff $\mathbf{P(B|A) = P(B)}$

Interpretation of Independence:

Observing *A* has no effect on probability of *B*. (and vice-versa)

Probability

Conditional Probability

$$P(A|B) = \frac{P(A, B)}{P(B)}$$

Independence example:

F1=H: first flip of a fair coin is heads
F2=H: second flip of the same coin is heads
$P(F1{=}H) = 0.5$ $P(F2{=}H) = \mathbf{0.5}$
$P(F2{=}H, F1{=}H) = 0.25$
$P(F2{=}H|F1{=}H) = \mathbf{0.5} = P(H2)$

Two events, A and B, are ***independent*** iff: $\mathbf{P(A, B) = P(A)P(B)}$

$\mathbf{P(A, B) = P(A)P(B)}$ iff $\mathbf{P(B|A) = P(B)}$

Interpretation of Independence:

Observing *A* has no effect on probability of *B*. (and vice-versa)

Probability

Conditional Probability

$$P(A|B) = \frac{P(A, B)}{P(B)}$$

Dependence example:

W1=happy: first word is "happy"
W2=birthday: second word is "birthday"

from observing language data, we find:

P(W1=happy) = 0.1, P(W2=birthday) = 0.05
P(W1=happy, W2=birthday) = 0.025

Two events, A and B, are ***independent*** iff: $P(A, B) = P(A)P(B)$

$P(A, B) = P(A)P(B)$ iff $P(B|A) = P(B)$

Interpretation of Independence:

Observing *A* has no effect on probability of *B*. (and vice-versa)

Probability

Conditional Probability

$$P(A|B) = \frac{P(A, B)}{P(B)}$$

Dependence example:

W1=happy: first word is "happy"
W2=birthday: second word is "birthday"

from observing language data, we find:

P(W1=happy) = 0.1, P(W2=birthday) = 0.05
P(W1=happy, W2=birthday) = 0.025

thus $P(A, B) \neq P(A)P(B)$
also $P(B|A) \neq P(B)$:
P(W2=birthday|W1=happy) = .025 / .1 = .25 ≠ 0.05 = P(W2=birthday)

Two events, A and B, are ***independent*** iff: $P(A, B) = P(A)P(B)$

$P(A, B) = P(A)P(B)$ iff $P(B|A) = P(B)$

Interpretation of Independence:

Observing *A* has no effect on probability of *B*. (and vice-versa)

Why Probability?

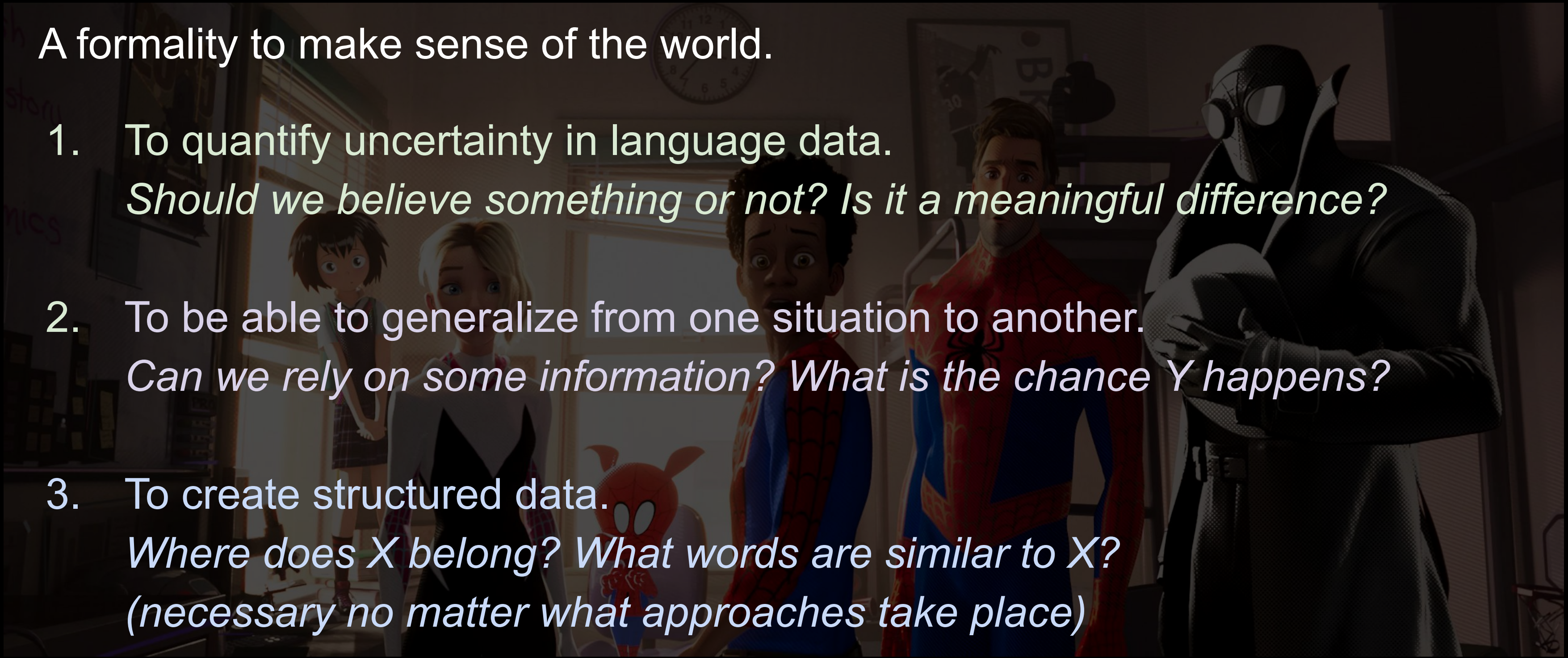
A formality to make sense of the world.

1. To quantify uncertainty in language data.
   Should we believe something or not? Is it a meaningful difference?

2. To be able to generalize from one situation to another.
   Can we rely on some information? What is the chance Y happens?

3. To create structured data.
   Where does X belong? What words are similar to X?
   (necessary no matter what approaches take place)

Why Probability?



A formality to make sense of the world.

1. To quantify uncertainty in language data.
   Should we believe something or not? Is it a meaningful difference?

2. To be able to generalize from one situation to another.
   Can we rely on some information? What is the chance Y happens?

3. To create structured data.
   Where does X belong? What words are similar to X?
   (necessary no matter what approaches take place)

Why Probability?

Words: Tokens and Types

word tokens - an individual word instance. (a list)

word types - distinct words. (a set)

V - "vocabulary" |***V***| - vocabulary size (number of types)

N - number of tokens

Words: Tokens and Types

word tokens - an individual word instance. (a list)

word types - distinct words. (a set)

V - "vocabulary" |***V***| - vocabulary size (number of types)

N - number of tokens

Corpus - a natural language dataset

(i.e. observational data of word sequence in the wild!)

Corpus	Tokens = N	Types = $|V|$
Shakespeare	884 thousand	31 thousand
Brown corpus	1 million	38 thousand
Switchboard telephone conversations	2.4 million	20 thousand
COCA	440 million	2 million
Google n-grams	1 trillion	13 million

Figure 2.11 Rough numbers of types and tokens for some English language corpora. The largest, the Google n-grams corpus, contains 13 million types, but this count only includes types appearing 40 or more times, so the true number would be much larger.

(SLP3, 2023)

V - "vocabulary" |***V***| - vocabulary size (number of types)

N - number of tokens

Corpus - a natural language dataset

(i.e. observational data of word sequence in the wild!)

Corpus	Tokens = N	Types = $|V|$
Shakespeare	884 thousand	31 thousand
Brown corpus	1 million	38 thousand
Switchboard telephone conversations	2.4 million	20 thousand
COCA	440 million	2 million
Google n-grams	1 trillion	13 million

Figure 2.11 Rough numbers of types and tokens for some English language corpora. The largest, the Google n-grams corpus, contains 13 million types, but this count only includes types appearing 40 or more times, so the true number would be much larger.

(SLP3, 2023)

V - "vocabulary" |***V***| - vocabulary size (number of types)

N - number of tokens

Corpus - a natural language dataset

(i.e. observational data of word sequence in the wild!)

Herndon or Heap's Law:

$$|V| = kN^{\beta}$$

Tokenizers

1.

2.

3.

Tokenizers

1. Word Tokenizers

2.

3.

Tokenizers

1. Word Tokenizers

```
import re
def tokenize(sentence):
    tokens = re.split(r'\s', sentence)
    return tokens
```

2.

3.

Tokenizers

1. Word Tokenizers

```
import re

def tokenize(sentence):

    tokens = re.split(r'\s', sentence)

    return tokens
```

   a. nltk's TreebankWordTokenizer

   b. DLATK's happierfuntokenizing.py (latest version)

2.

Tokenizers

1. Word Tokenizers

2. Byte-Pair Encoding

3.

Tokenizers

1. Word Tokenizers

2. Byte-Pair Encoding
   Motivations:
   – more data-driven; no predefined words or rules
   – allow for *subwords* (e.g. "unlikeliest" -> "un", "like", "liest") – better for unseen words or capturing semantics of parts of words.

3.

Tokenizers

1. Word Tokenizers

2. Byte-Pair Encoding

   Motivations:

   – more data-driven; no pre

   – allow for *subwords* (e.g. '

   unseen words or capturing

3.

```
Input: set of strings D, target vocab size k
procedure BPE(D, k)
    V ← all unique characters in D
            (about 4,000 in English Wikipedia)
    while |V| < k do                          ▷ Merge tokens
        t_L, t_R ← Most frequent bigram in D
        t_NEW ← t_L + t_R                     ▷ Make new token
        V ← V + [t_NEW]
        Replace each occurrence of t_L, t_R in
            D with t_NEW
    end while
    return V
end procedure
```

(Bostrum & Durrett, 2020)

Tokenizers

1. Word Tokenizers

2. Byte-Pair Encoding

   Motivations:

   – more data-driven; no predefined w

   – allow for *subwords* (e.g. "unlikelies

   unseen words or capturing semantic

```
Input: set of strings D, target vocab size k
procedure BPE(D, k)
    V ← all unique characters in D
            (about 4,000 in English Wikipedia)
    while |V| < k do                          ▷ Merge tokens
        t_L, t_R ← Most frequent bigram in D
        t_NEW ← t_L + t_R                     ▷ Make new token
        V ← V + [t_NEW]
        Replace each occurrence of t_L, t_R in
            D with t_NEW
    end while
    return V
end procedure
```

([Bostrum & Durrett, 2020](#))

corpus

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

vocabulary

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

(SLP3, p.18)

Tokenizers

1. Word Tokenizers

2. Byte-Pair Encoding

   Motivations:

   – more data-driven; no predefined w

   – allow for *subwords* (e.g. "unlikelies

   unseen words or capturing semantic

Input: set of strings D, target vocab size k
procedure BPE(D, k)
  $V \leftarrow$ all unique characters in D
    (about 4,000 in English Wikipedia)
  while $|V| < k$ **do** ▷ Merge tokens
    $t_L, t_R \leftarrow$ Most frequent bigram in D
    $t_{\text{NEW}} \leftarrow t_L + t_R$ ▷ Make new token
    $V \leftarrow V + [t_{\text{NEW}}]$
    Replace each occurrence of t_L, t_R in
      D with t_{NEW}
  end while
  return V
end procedure

([Bostrum & Durrett, 2020](#))

corpus

l o w _	l o w e s t _	ne w er_	w i d er_
l o w _	l o w e s t _	ne w er_	ne w _
l o w _	ne w er_	ne w er_	ne w _
l o w _	ne w er_	w i d er_	
l o w _	ne w e r _	w i d er_	

vocabulary (original)

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

vocabulary (3 iterations later)

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

(SLP3, p.19)

Tokenizers

1. Word Tokenizers

2. Byte-Pair Encoding

3. Wordpiece

   Choose pairings based on what increases likelihood of data.

   Does putting "a" and "b" together increase ability to model the corpus?

   This can be quantified by: $\dfrac{p('ab')}{p('a')p('b')}$

   More here: (Shuster and Nakajima, 2012; Kudo and Richardson, 2018)