Natural Language Processing: Introduction and Preliminaries

CSE354 - Spring 2021

Instructor: H. Andrew Schwartz

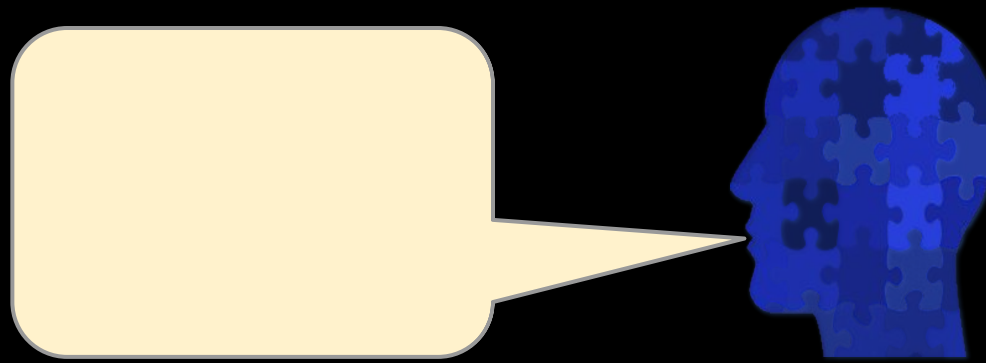
1. General goal for NLP and appreciation for complexity.

2. Course Overview

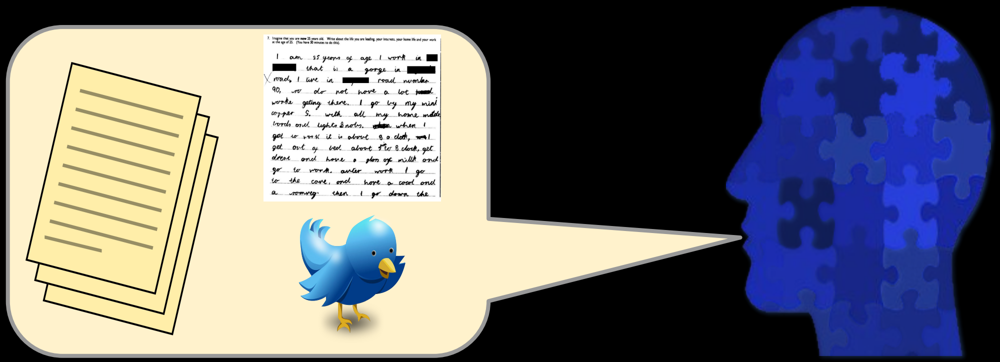
3. Preliminary methods
   a. Regular Expressions
   b. Probability Theory

Natural language is complicated!

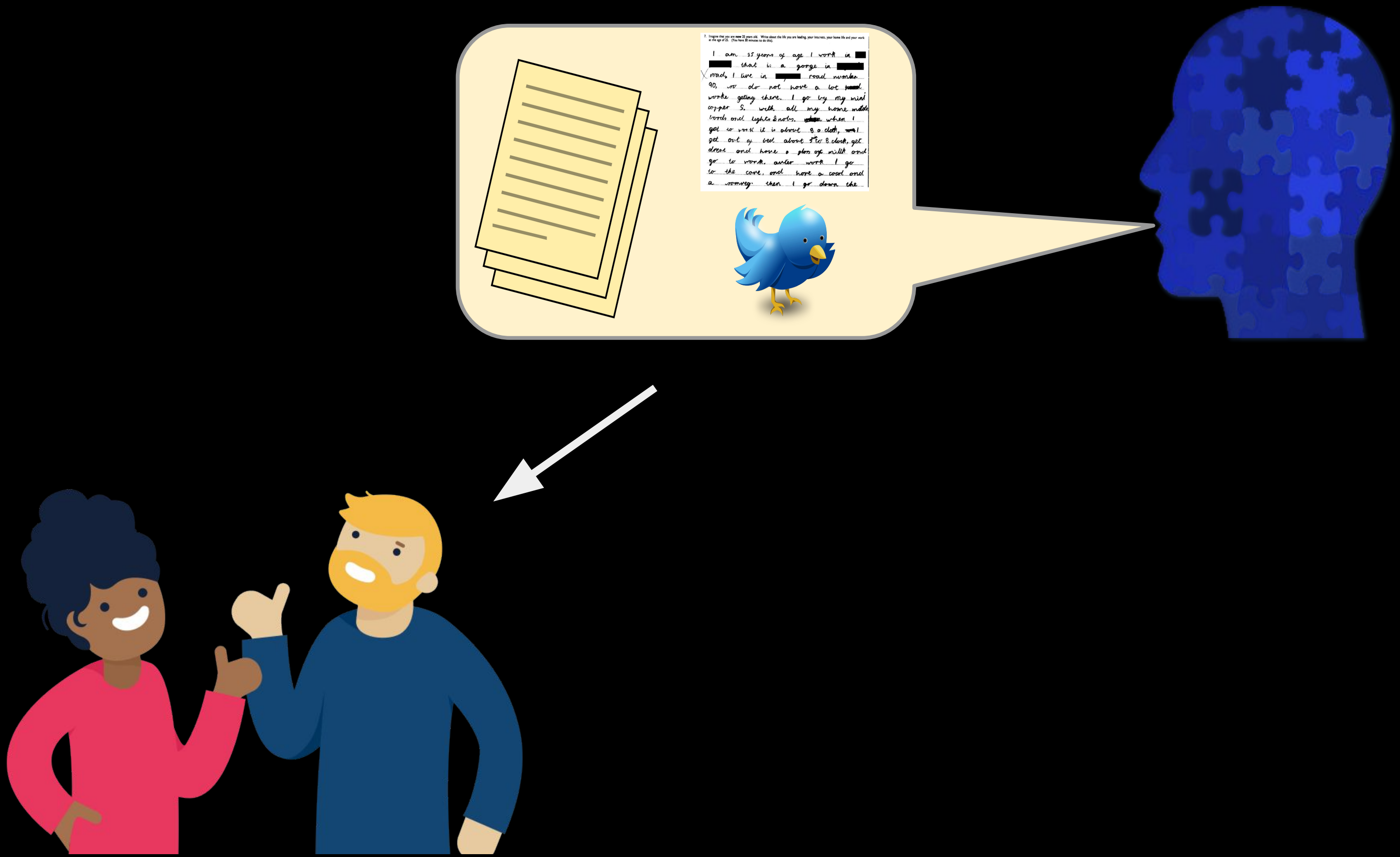
Natural language is complicated!



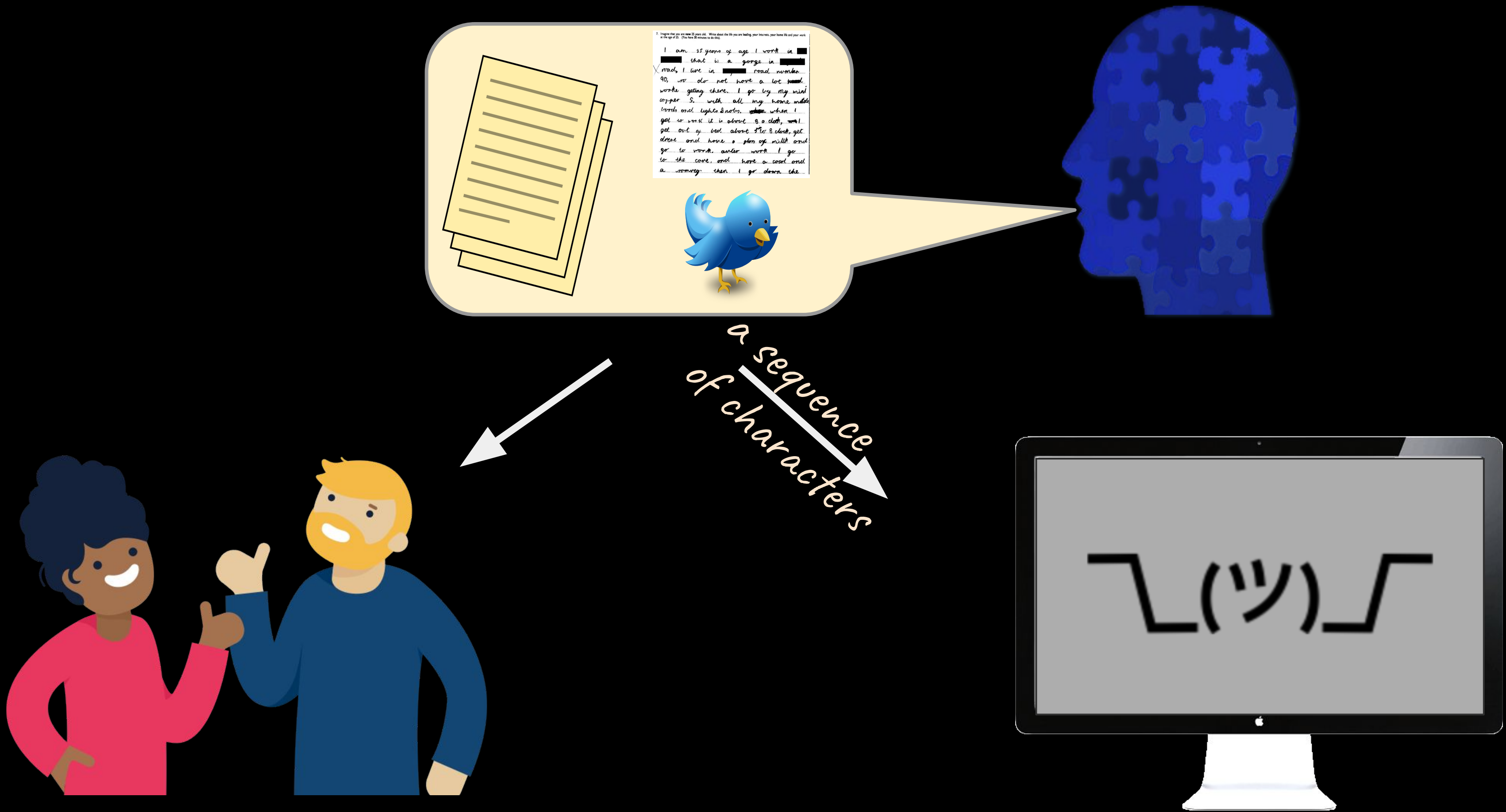
Natural language is complicated!



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. ✓

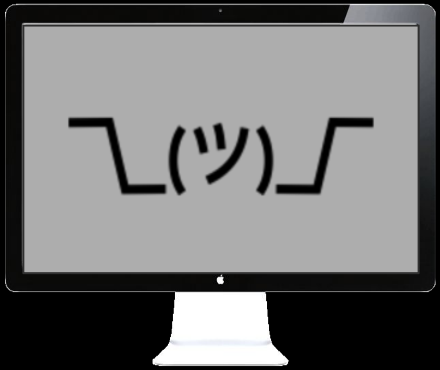
The horse **raced** past the barn fell. ✓

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...

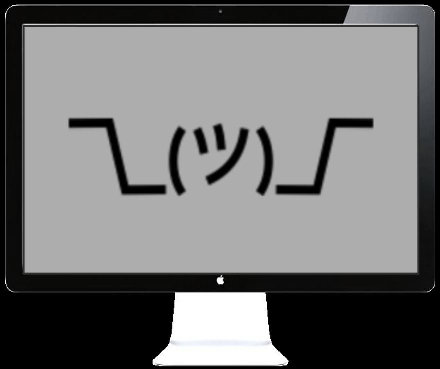
Colorless purple ideas sleep furiously. (Chomsky, 1956; “purple”=> “green”)

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...



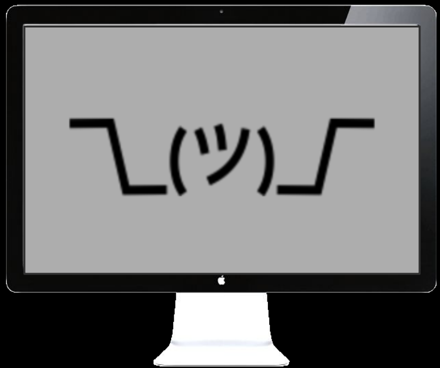
She ate the cake with the frosting.

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.

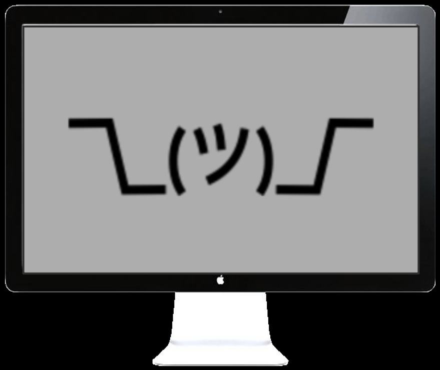
```
['She', 'ate', X, 'with', Y, '.']
=> Y is a part of X
```

More empathy for the computer...

She ate the cake with the frosting.

She ate the cake with the fork.

More empathy for the computer...

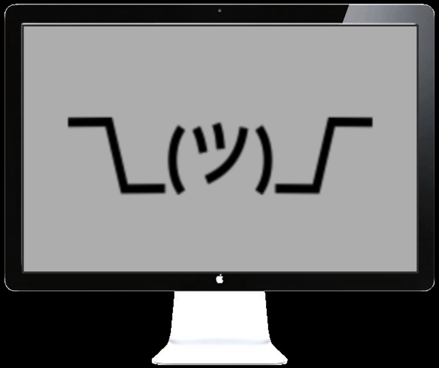


She ate the cake with the frosting.

She ate the cake with the fork.

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

More empathy for the computer...



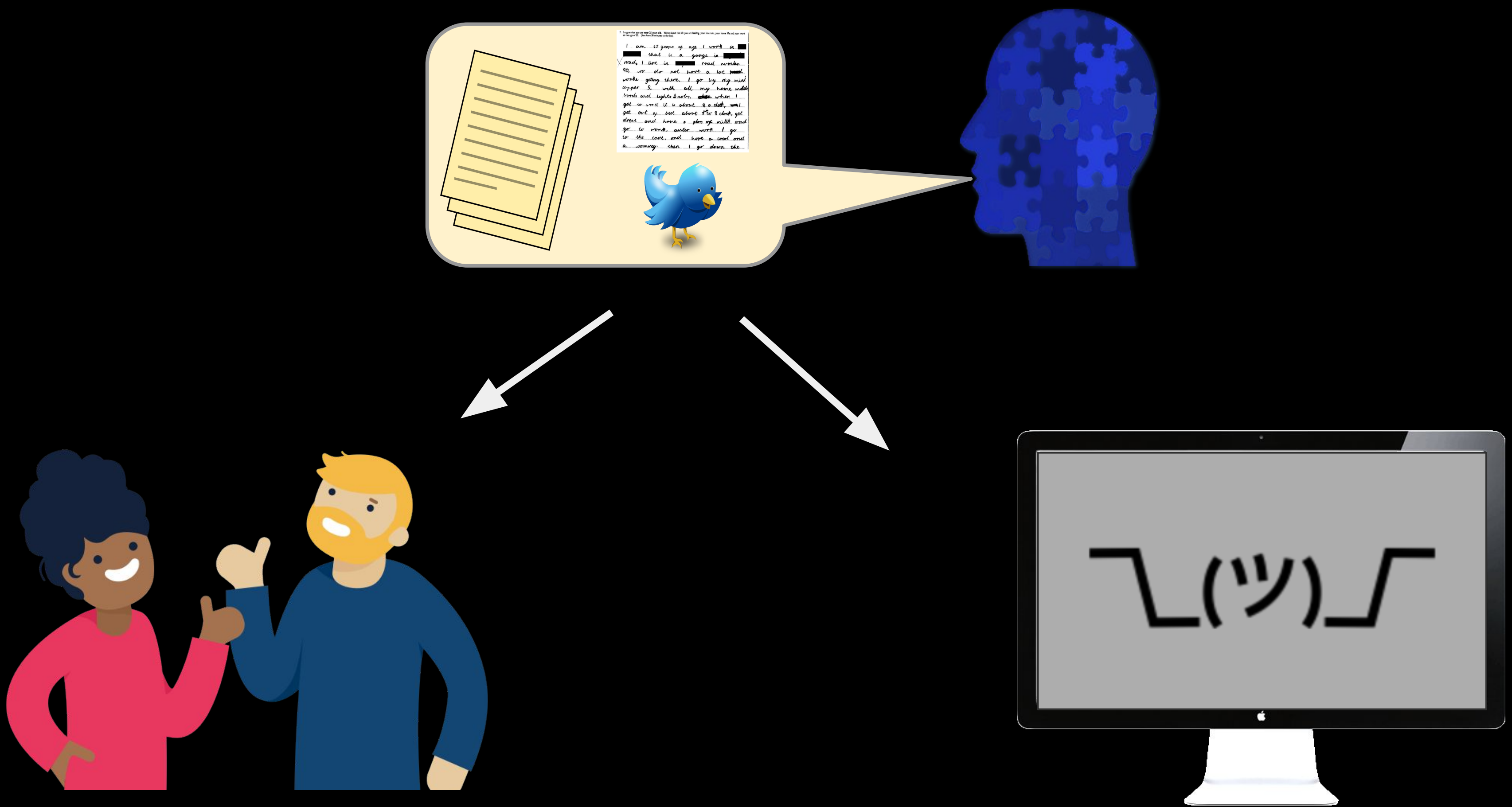
She ate the cake with the frosting.

She ate the cake with the fork.

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

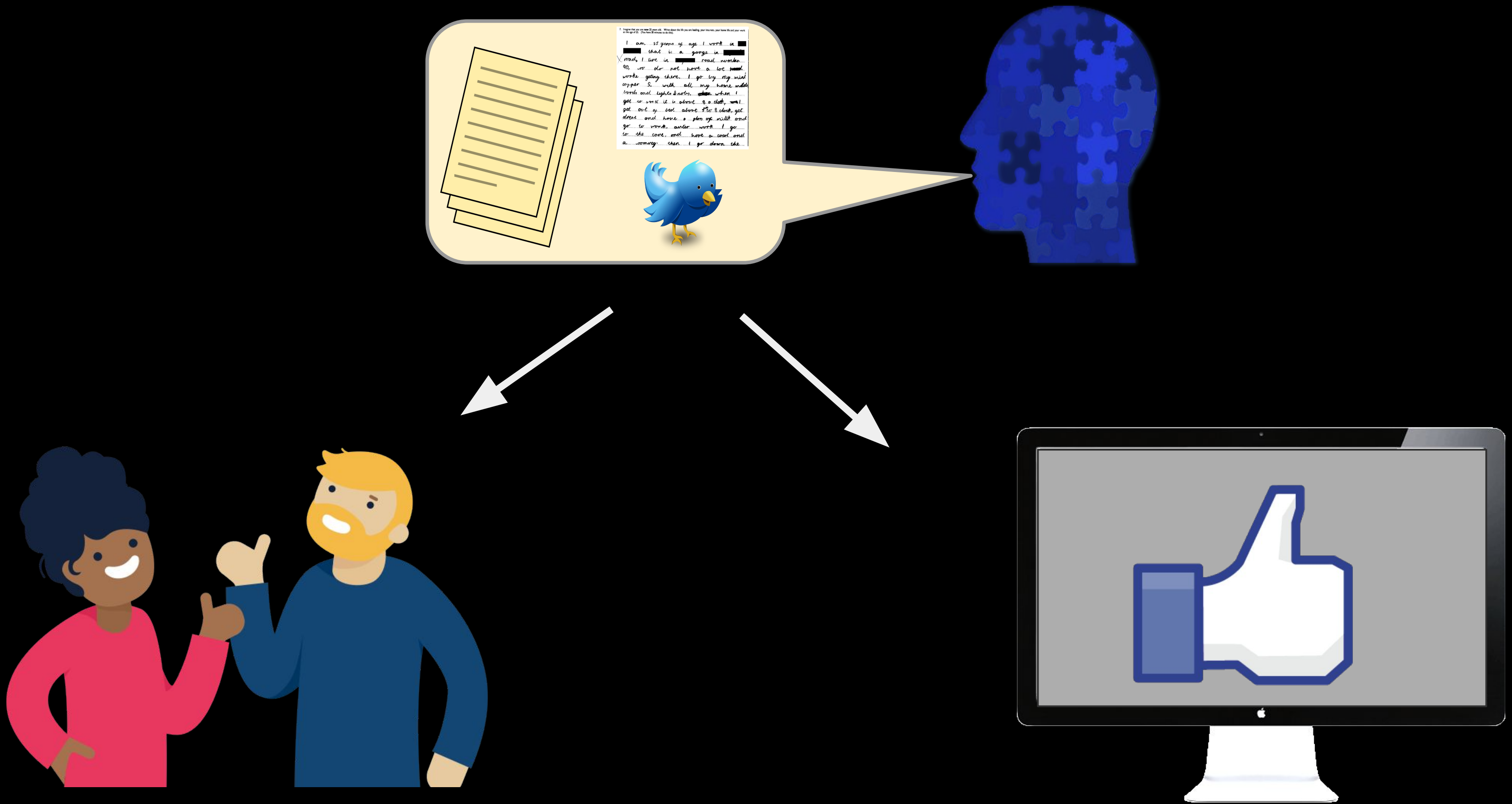
He walked along the **port** of the ship.

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

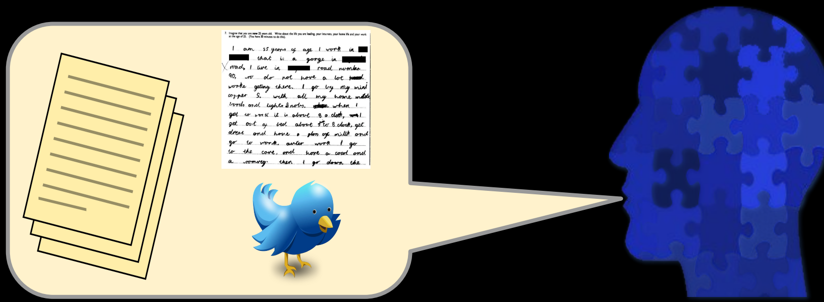


¯_(ツ)_/¯

NLP’s grand goal: completely understand natural language.

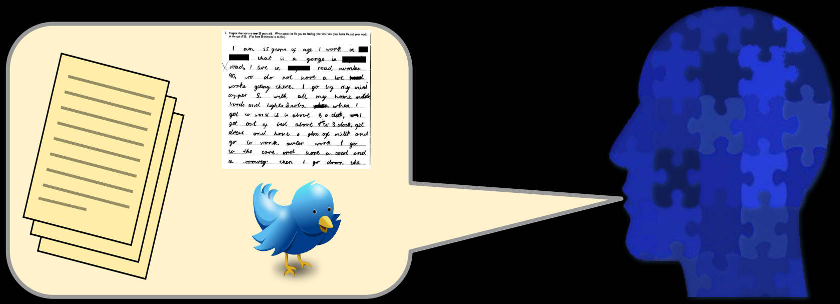


NLP’s practical applications



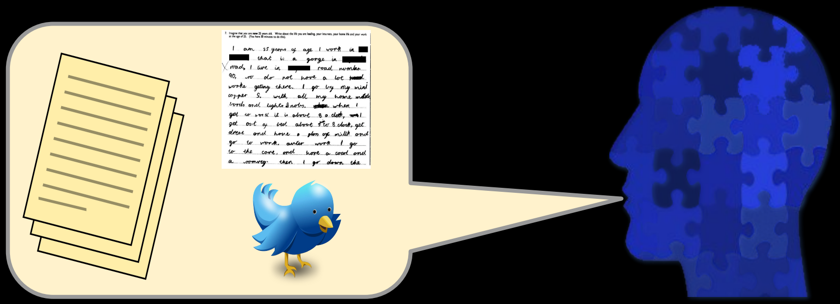
- Machine translation

NLP's practical applications



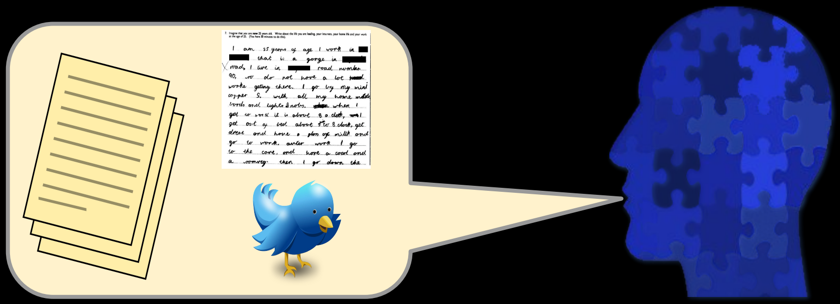
- Machine translation
- Sentiment Analysis

NLP’s practical applications



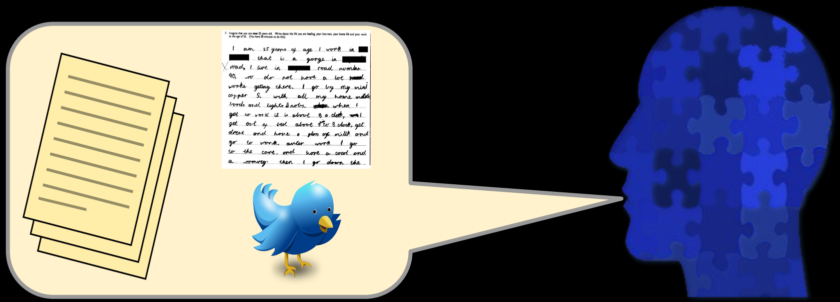
- Machine translation
- Sentiment Analysis
- Automatic speech recognition
  - Personalized assistants
  - Auto customer service

NLP’s practical applications



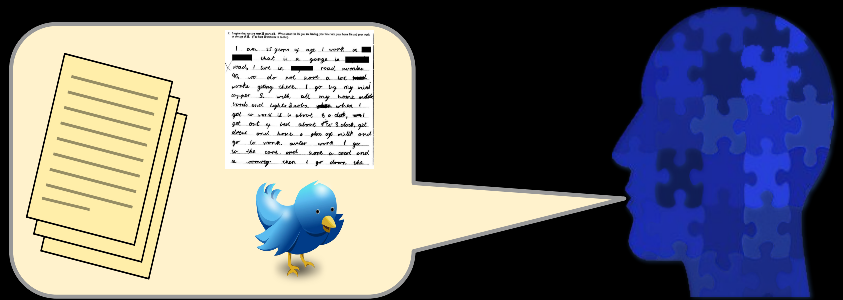
- Machine translation
- Sentiment Analysis
- Automatic speech recognition
  - Personalized assistants
  - Auto customer service
- Information Retrieval
  - Web Search
  - Question Answering

NLP's practical applications



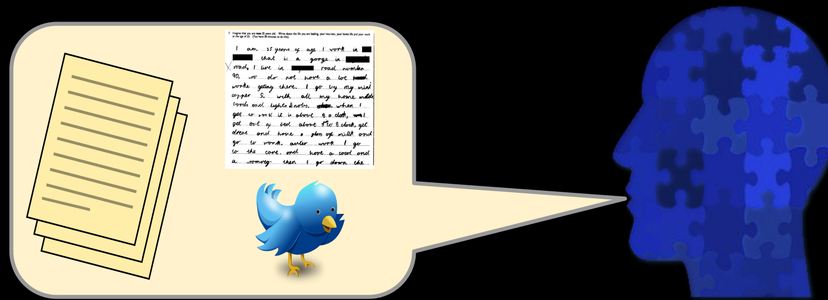
- Machine translation
- Sentiment Analysis
- Automatic speech recognition
  - Personalized assistants
  - Auto customer service
- Information Retrieval
  - Web Search
  - Question Answering
- Computational Social Science

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
- *Growing day by day*

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
- *Growing day by day*

how? →

- Machine learning:
  - Logistic regression
  - Probabilistic modeling
  - Recurrent Neural Networks
  - Transformers
- Algorithms, e.g.:
  - Graph analytics
  - Dynamic programming
- Data science
  - Hypothesis testing

NLP: The Coarse

Speech and Language Processing

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

Third Edition draft

Daniel Jurafsky
Stanford University

James H. Martin
University of Colorado at Boulder

Draft of December 30, 2020. Comments and typos welcome!

web.stanford.edu/~jurafsky/slp3/

Course Website - Syllabus

www3.cs.stonybrook.edu/~has/CSE354/

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.

- ➔ **Readings:** 2 hours/wk; 10 - 20 pages/wk (best before each class)
- ➔ **Study:** 1 - 2 hours/wk to review notes and look up extra content
  (plus 3 to 4 hours to review before final exam)
- ➔ **Homeworks (4):** 5 to 8 hours each
- ➔ **NLP in the World (1):** 5 to 8 hours preparing presentation

Preliminary Methods

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?

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<u>**ing**</u>', '<u>**ing**</u>les', 'class'X
[sS]bu	'sbu', 'I like Sbu a lot', 'SBU'	'<u>**sbu**</u>', 'I like <u>**Sbu**</u> a lot', 'SBU'X
[A-Z][a-z]	'sbu', 'Sbu' #capital followed by lowercase	'sbu'X, '<u>**Sb**</u>u'
[0-9][MmKk]	'5m', '50m', '2k', '2b'	'<u>**5m**</u>', '5<u>**0m**</u>', '<u>**2k**</u>', '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 ’	

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

Patte

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

<table>
<tr><th>pattern</th><th>example strings</th><th>matches</th></tr>
<tr><td>r’hers|his’</td><td>‘this is hers’, ‘this is his!’</td><td>‘this is <u>hers</u>’, ‘this is <u>his</u>!’</td></tr>
<tr><td>r’([A-Z][a-z]+)+’</td><td>‘This matches Cap Words followed By a Space.’</td><td>‘<u>This </u>matches <u>Cap Words </u>followed <u>By </u>a Space.’</td></tr>
</table>

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’	‘**<u>k</u>**’ ‘**<u>i</u>**’ ‘**<u>c</u>**’ ‘**<u>k</u>**’
.$	‘great’, ‘great!’, ‘50’	‘grea**<u>t</u>**’, ‘great**<u>!</u>**’, ‘5**<u>0</u>**’

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:

<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></td></tr>
</table>

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>
</table>

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

```python
import re

words = re.findall(r'\b[A-z]+\b', 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>

Regular Expressions

```
import re
words = re.split(r'\s', sentence)
for word in words:
    print(word)
```

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

Probability

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?

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

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

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 $A_1, A_2, \ldots$ are disjoint events then: $$P\left(\bigcup_i^\infty A_i\right) = \sum_i^\infty P(A_i)$$

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

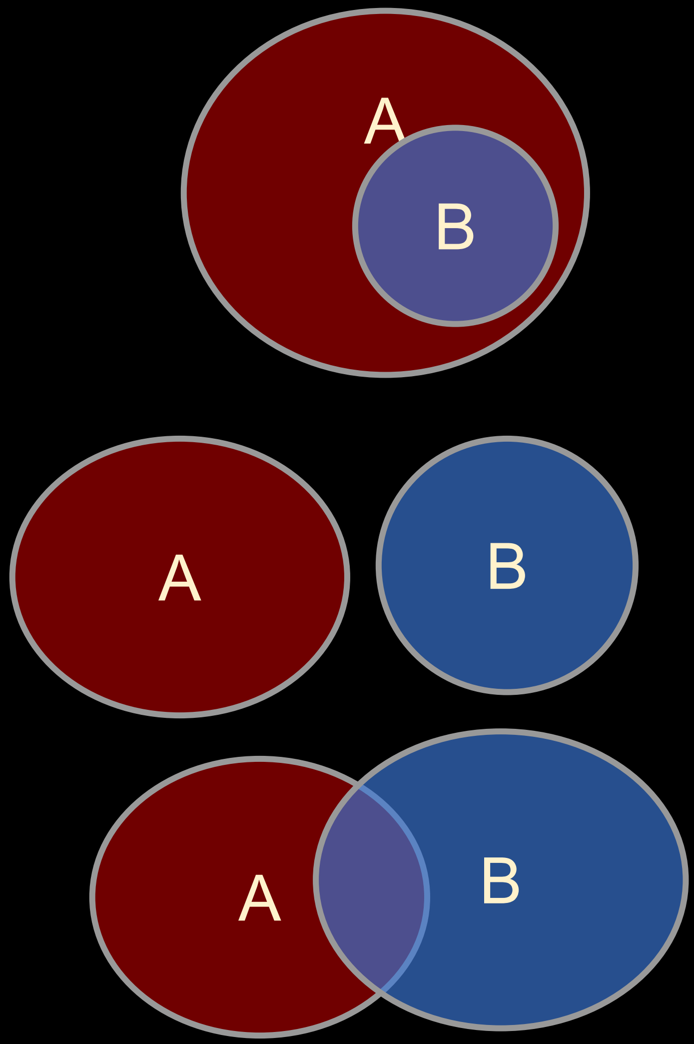
P is a *probability measure*, if and only if

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

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

Probability

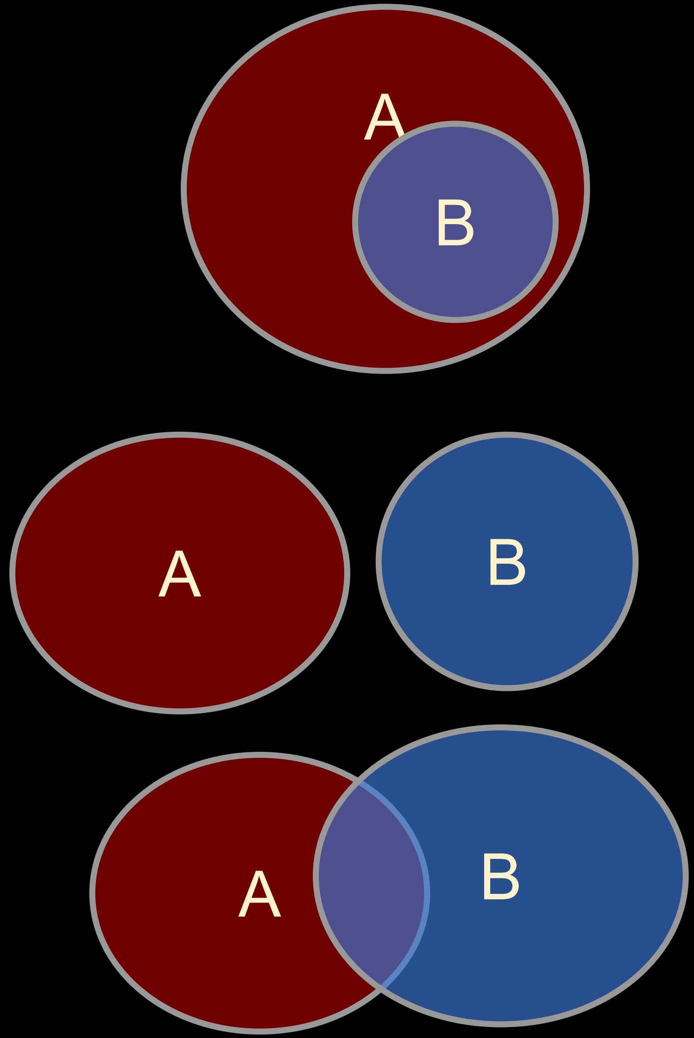
Some Properties:



Probability

Some Properties:

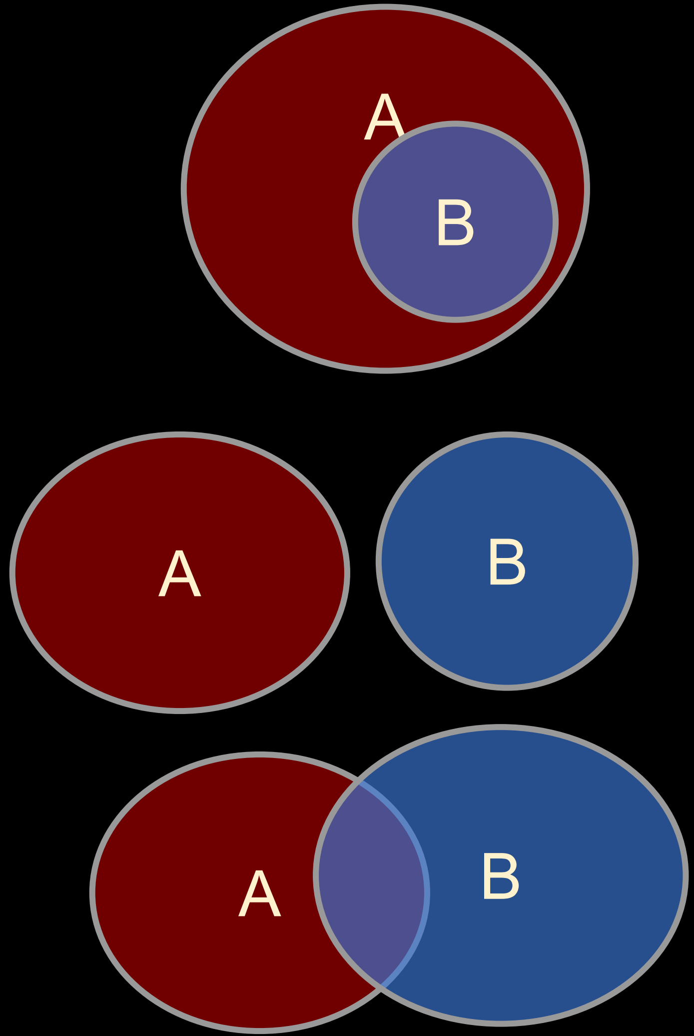
1. If $B \subseteq A$ then $\mathbf{P}(A) \geq \mathbf{P}(B)$



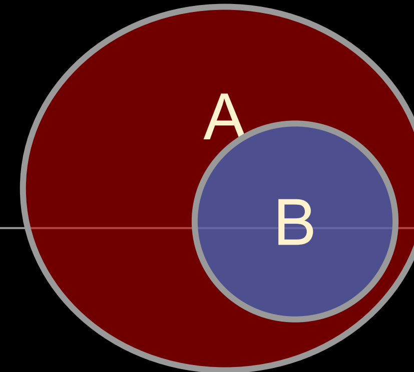
Probability

Some Properties:

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



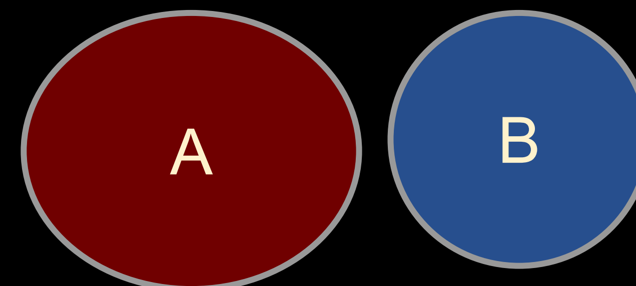
Probability

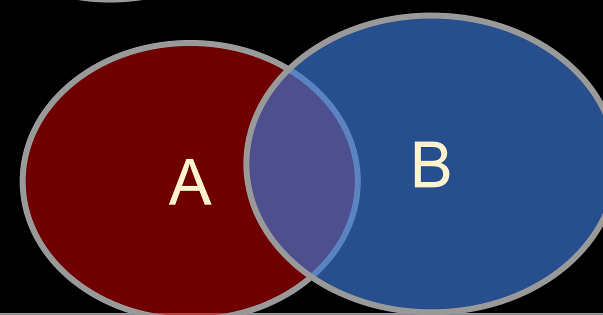


Some Properties:

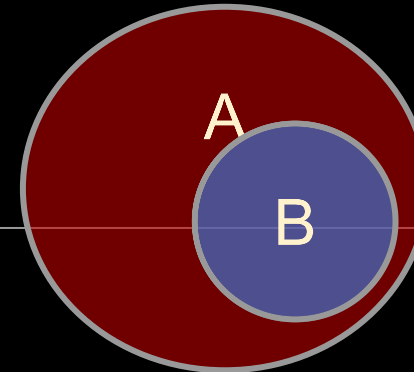
1. If $B \subseteq A$ then $\mathbf{P}(A) \geq \mathbf{P}(B)$
2. $\mathbf{P}(A \cup B) \leq \mathbf{P}(A) + \mathbf{P}(B)$
3. $\mathbf{P}(A \cap B) \leq \mathbf{min}(\mathbf{P}(A), \mathbf{P}(B))$
4. $\mathbf{P}(\neg A) = \mathbf{P}(\Omega \,/\, A) = \mathbf{1} - \mathbf{P}(A)$

/ is set difference





Probability

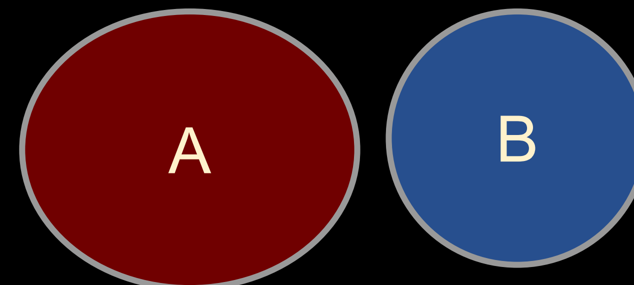


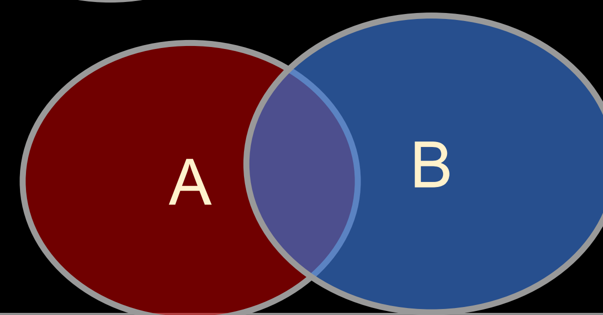
Some Properties:

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: $\mathbf{P(\mathit{A}, \mathit{B}) = P(\mathit{A})P(\mathit{B})}$

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* <u>has no effect on</u> probability of *B.*

(Disjoint events, typically, are <u>not</u> 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: **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

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)