Natural Language Processing: Introduction and Preliminaries

Frankfurt Sept. 2024
Material:.Stonybrook, Stanford,
ETH-Z and others

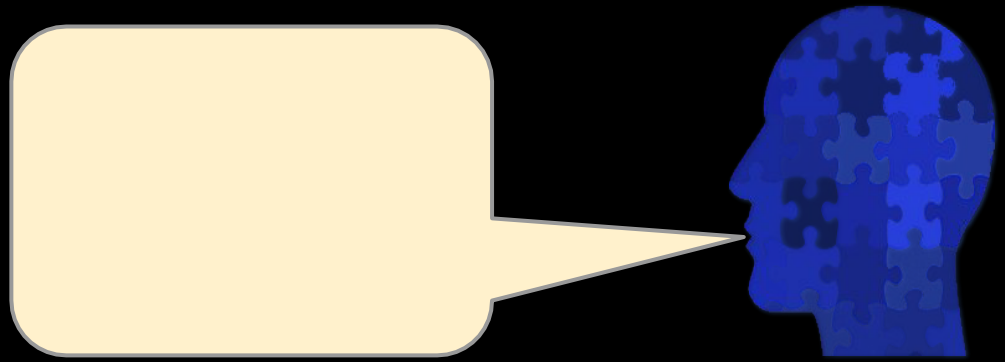
1. General goal for NLP and appreciation for complexity.

2. Course Overview

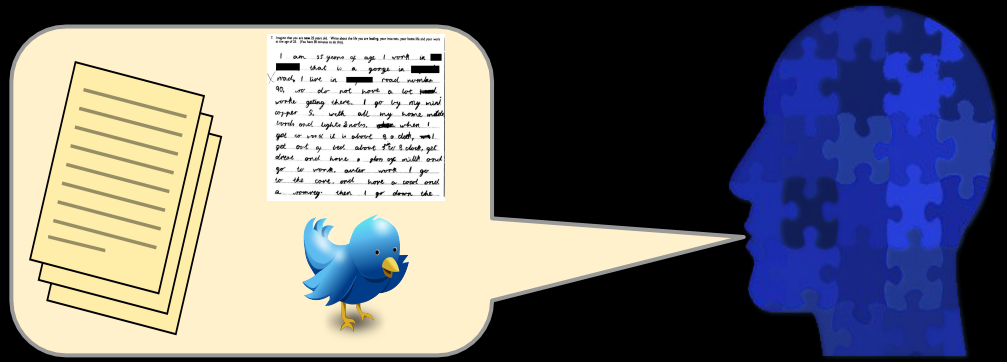
3. Preliminary methods
    a. Regular Expressions
    b. Probability Theory
    c. N-grams

Natural language is complicated!

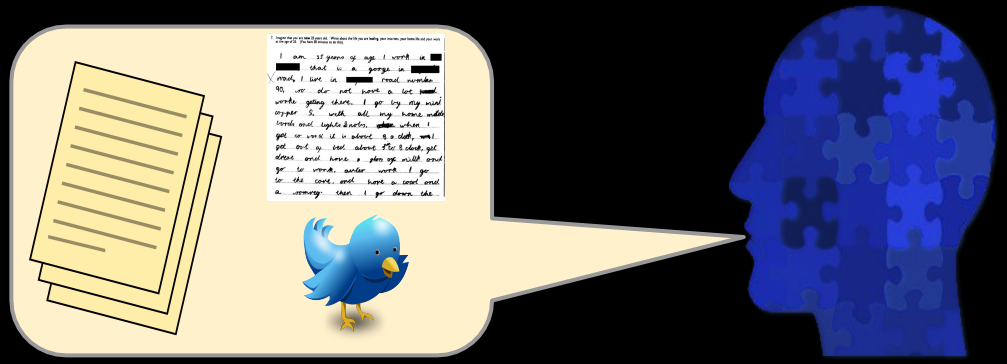
Natural language is complicated!



Natural language is complicated!



Natural language is complicated!



Natural language is complicated!

a sequence of characters

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. ✓

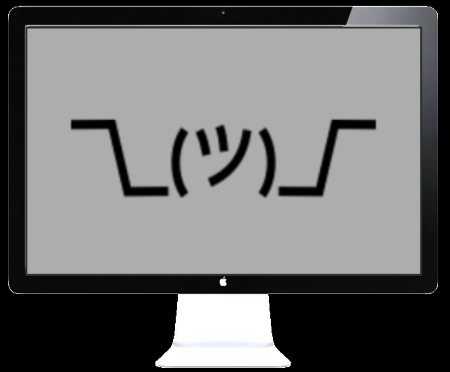
The horse **raced** past the barn fell. ✓

that was

The horse **runs** past the barn. ✓

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

More empathy for the computer...



Colorless green ideas sleep furiously. (Chomsky, 1956;)

More empathy for the computer...

Colorless green ideas sleep furiously. (Chomsky, 1956;)

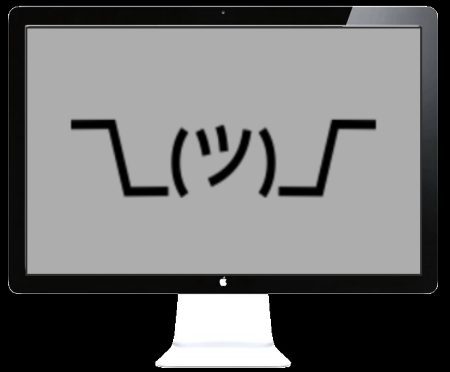
Fruit flies like a banana. Time flies like an arrow.

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...

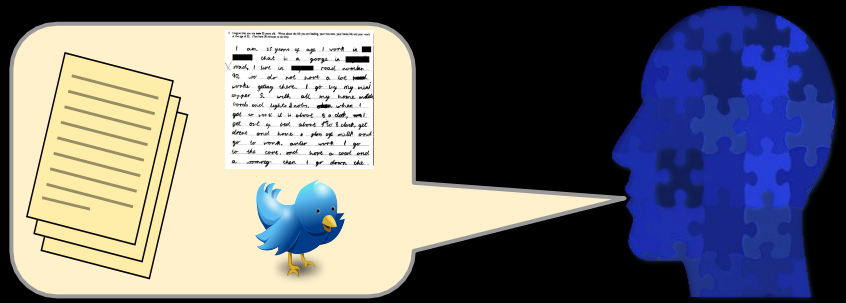
She ate the cake with the frosting.

She ate the cake with the fork.

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

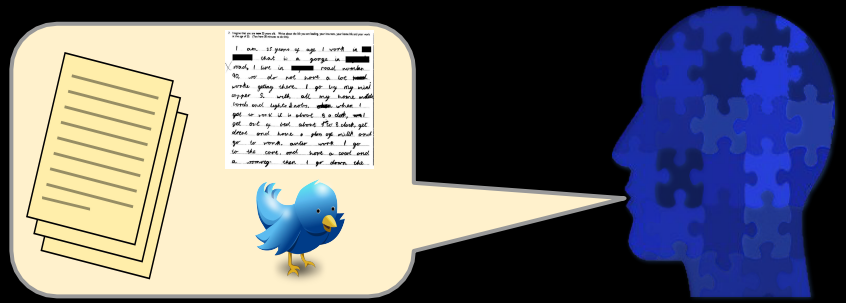
He walked along the **port** of the ship.

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

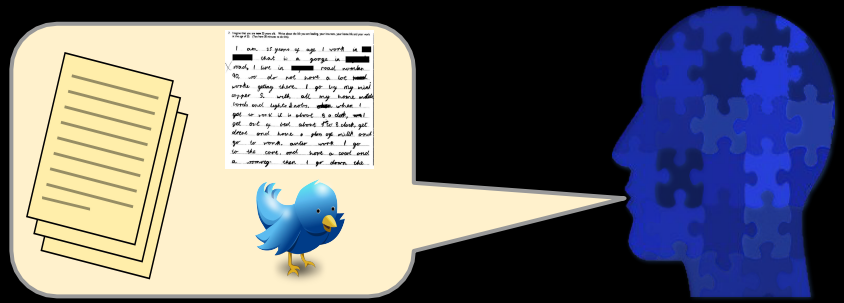



¯_(ツ)_/¯

NLP's grand goal: completely understand natural language.

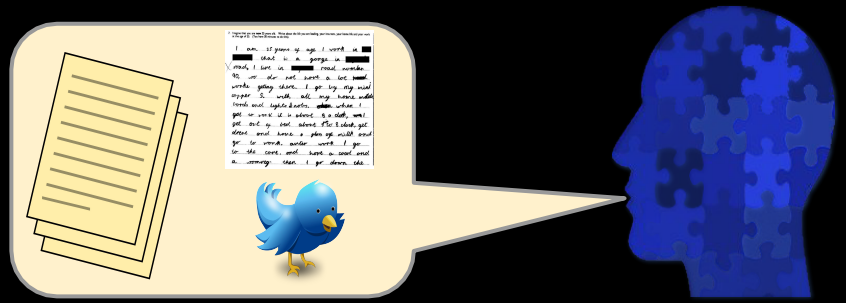


NLP’s practical applications



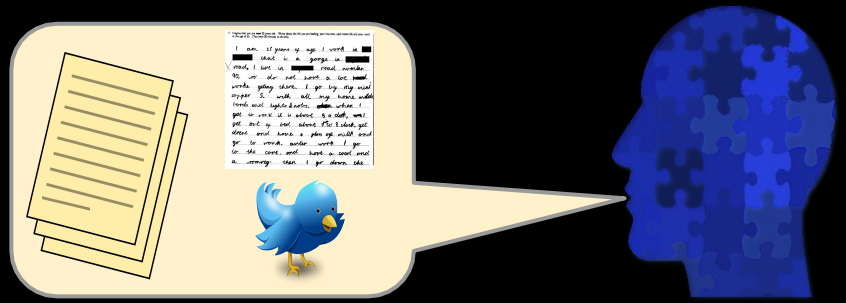
- Machine translation

NLP’s practical applications



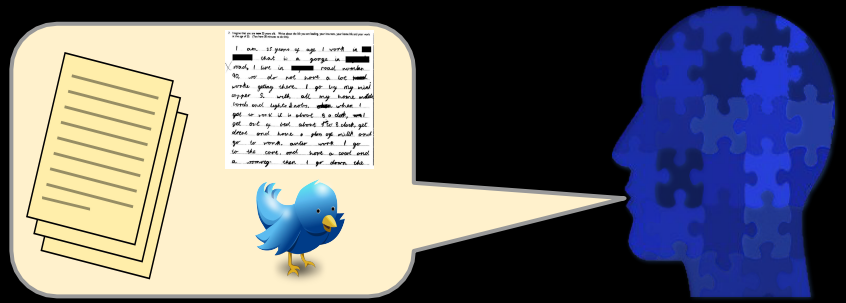
- Machine translation
- Sentiment Analysis

NLP's practical applications



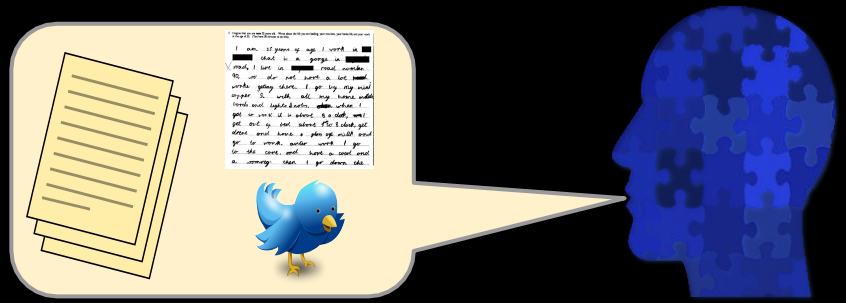
- Machine translation
- Sentiment Analysis
- Automatic speech recognition
  - Personalized assistants
  - Auto customer service

NLP's practical applications



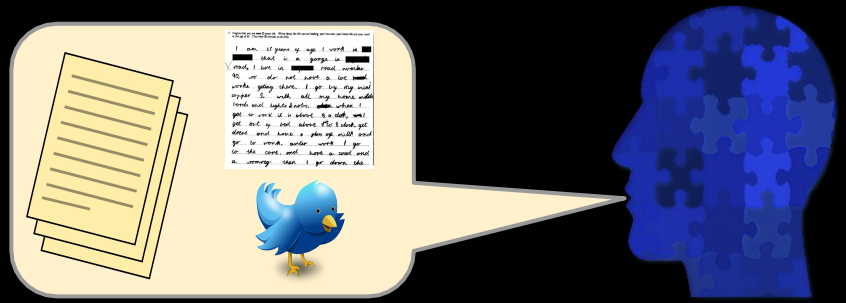
- Machine translation
- Sentiment Analysis
- Automatic speech recognition
  - Personalized assistants
  - Auto customer service
- Information Retrieval
  - Web Search
  - Question Answering

NLP's practical applications



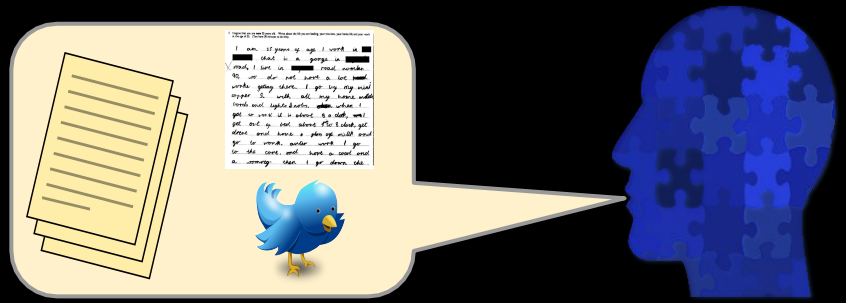
- Machine translation
- Sentiment Analysis
- Automatic speech recognition
  - Personalized assistants
  - Auto customer service
- Information Retrieval
  - Web Search
  - Question Answering
- Computational Social Science

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 Course

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/

Ingredients for success

The following covers the major components of the course..

- ➔ **Intensive 2 days x 8 hours per day. 1 half day (Thursday), half-day, homework (after course) followed by online session(s).**
- ➔ **Lectures/Discussions: ~3 hours/day**
- ➔ **Labs: 3-4 hours/day**
- ➔ **Project: T.B.A.**

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?

A. Regular Expressions

An introduction to regular expressions (and their use in Python)

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 ’	

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

Patterns to match in a string.

In python we denote regular expressions with:

r'PATTERN'

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
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**', '**orange**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:

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

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.'	'**Kick**' ' **a**' ' **door.**'
r'\b[A-z]+\b'	'Kick a door.'	'**Kick** **a** **door**.' #3 matches, no whitespace

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

Lab01: Regular Expressions (Python3 Jupyter Notebook)

Goal:
1) Warm-up
2) Make sure that conda environments, the Jupyter notebook and Python3 kernels are correctly installed
3) Refresh a bit of Python and regular expressions

Regular Expressions are everywhere
(as a general pattern matcher).

Widely used in both academics and industry

1. Part of most text processing tasks, even for big neural language model pipelines,
2. including text formatting and pre-processing
3. Very useful for data analysis of any text data

Early Chatbot (ELIZA, Joseph Weizenbaum, 1966)

Its DOCTOR script was an early NLP system that imitated a Rogerian psychotherapist

Uses pattern matching to match, e.g.,:

- `"I need X"`

and translates them into, e.g.

- `"What would it mean to you if you got X?`

Early Chatbot (ELIZA, Joseph Weizenbaum, 1966)

Men are all alike.
IN WHAT WAY

They're always bugging us about something or other.
CAN YOU THINK OF A SPECIFIC EXAMPLE

Well, my boyfriend made me come here.
YOUR BOYFRIEND MADE YOU COME HERE

He says I'm depressed much of the time.
I AM SORRY TO HEAR YOU ARE DEPRESSED

How ELIZA (Doctor Script) works

The `s` command (as in substitute) is probably the most important in `sed` and has a lot of different options. The syntax of the `s` command is '`s/`*`regexp/replacement/flags`*'.

(It was actually written in MAD-SLIP. Symmetric List Process as a Lisp-like extension to Fortran)

s/.* I'M (depressed|sad) .*/I AM SORRY TO HEAR YOU ARE \1/

s/.* I AM (depressed|sad) .*/WHY DO YOU THINK YOU ARE \1/

s/.* all .*/IN WHAT WAY?/

s/.* always .*/CAN YOU THINK OF A SPECIFIC EXAMPLE?/

```
  Welcome to
                    EEEEEE  LL      IIII   ZZZZZZ   AAAAA
                    EE      LL       II       ZZ   AA   AA
                    EEEEE   LL       II     ZZZ    AAAAAAA
                    EE      LL       II    ZZ      AA   AA
                    EEEEEE  LLLLLL  IIII ZZZZZZ    AA   AA

  Eliza is a mock Rogerian psychotherapist.
  The original program was described by Joseph Weizenbaum in 1966.
  This implementation by Norbert Landsteiner 2005.


ELIZA: Is something troubling you ?
YOU:   Men are all alike.
ELIZA: What is the connection, do you suppose ?
YOU:   They're always bugging us about something or other.
ELIZA: Can you think of a specific example ?
YOU:   Well, my boyfriend made me come here.
ELIZA: Is it important to you that your boyfriend made you come here ?
YOU:   He says I'm depressed much of the time.
ELIZA: I am sorry to hear that you are depressed.
YOU:   It's true. I am unhappy.
ELIZA: Can you explain what made you unhappy ?
YOU:
```

From ELIZA 1966 to ChatGPT/Gemini/LLaMa/..

Basically instead of comparatively “primitive” regular expressions of Eliza these modern LLM-ChatBots use attention models (which we’ll talk about in later lectures) to predict and match. They can do this by learning word order (they don’t UNDERSTAND anything) by studying massive amounts of text.

Including:

- Common Crawl (https://commoncrawl.org/) which contains over 250 billion pages spanning 17 years.

(As an aside: https://openwebsearch.eu/ is an EU initiative supported by the EU NGI to develop its own crawl or search, AI etc)

Online (for the curious)

https://web.njit.edu/~ronkowit/eliza.html

(There are literally 100s of people running online versions)

Peabody Award: https://peabodyawards.com/award-profile/eliza-1964/

NLP Libraries

We need to start off with tokenisation.

Popular Python Libs:

1) NTLK: https://www.nltk.org/
Its been long the most popular NLP Toolkit for R&D/Teaching.

2) SpaCy: https://spacy.io/usage/spacy-101
In contrast to NTLK its focus is on providing software for production usage. Its designed for larger amounts of data and performance.

3) Hugging Face Transformers

Tokenisation

It is a bit more complicated than just white spaces. What about things like hyphenation? Or word contractions like “can’t” or “don’t,” where the meaning is often dependent on the combination of words?

Tokenisation

Can't just **blindly** remove punctuation:

- m.p.h., Ph.D., AT&T, cap’n
- prices ($45.55)
- dates (01/02/06)
- URLs (http://www.stanford.edu)
- hashtags (#nlproc)
- email addresses (someone@cs.colorado.edu)

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

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

When should multiword expressions (MWE) be words?

- New York, rock ’n’ roll

Tokenisation

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

How do we decide where the token boundaries should be?

(Beyond the scope of this course but deciding what counts as a word in Chinese is complex and not agreed upon.)

Tokenisation

In re-Isearch we preserve hyphenated words BUT have a search algorithm that matches: auto-mobile == automobile. We can match “auto mobile” to “auto-mobile” in text but not “automobile” as the concatenation is not always semantically equivalent. The other direction is also not always the case. Example: “head-hunter” versus “head hunter”.

In a entity recognition, however, we have other issues: Example: Must-have, hunter-gatherer, Prinz-Albrecht-Palais

(Detour) Subword tokenisation

Three common algorithms:

- **Byte-Pair Encoding (BPE)** (Sennrich et al., 2016)
- **Unigram language modeling tokenization** (Kudo, 2018)
- **WordPiece** (Schuster and Nakajima, 2012)

All have 2 parts:

- A token **learner** that takes a raw training corpus and induces a vocabulary (a set of tokens).
- A token **segmenter** that takes a raw test sentence and tokenizes it according to that vocabulary

Word Normalisation

Putting words/tokens in a standard format

- U.S.A. or USA
- uhhuh or uh-huh
- Fed or fed
- am, is, be, are

Case Folding

Applications like IR: reduce all letters to lower case

- Since users tend to use lower case
- Possible exception: upper case in mid-sentence?
  - e.g., ***General Motors***
  - ***Fed*** vs. ***fed***
  - ***SAIL*** vs. ***sail***

For sentiment analysis, MT, Information extraction

- Case is helpful (***US*** versus ***us*** is important)

Case Folding

NOTE: Not all languages have simple case maps for glyphs. This is simple for English but can be quite difficult in Unicode since it is based on glyphs and not letters.

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

Stemming

Porter's Algorithm (computationally simple)

Based on a series of rewrite rules run in series
A cascade, in which output of each pass fed to next pass

Some sample rules:

ATIONAL → ATE (e.g., relational → relate)
ING → ϵ if stem contains vowel (e.g.,

Lemmatization

Represent all words as their lemma, their shared root

= dictionary headword form:

- *am, are, is → be*
- *car, cars, car's, cars' → car*
- Spanish quiero (‘I want’), quieres (‘you want’) → querer ‘want’
- *He is reading detective stories → He be read detective story*

Lemmatization

Lemmatization is done by Morphological Parsing

Morphemes:

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

Lemmatization

Morphological Parsers:

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

Why not just use stemming?

Dealing with complex morphology is necessary for many languages (e.g. German!)

Original vs Stemmed vs Lemmatised

Original **Kim arrived on Tuesday morning at the border station of Dong Dang and was welcomed with a military salute reminiscent of the fanfare that greeted his grandfather Kim Il-sung more than half a century ago. His heavily armoured train pulled into the station just after 8am, after a 65-hour, 4,500km journey from Pyongyang via mainland China. Just as they did during Kim Il-sung's visit 55 years ago, a dozen Vietnamese soldiers in white and green uniforms saluted the 35-year-old leader as he emerged from the train and walked on the red carpet.**

Stemmed kim arriv on tuesday morn at the border station of dong dang and wa welcom with a militari salut reminisc of the fanfar that greet hi grandfath kim il-sung more than half a centuri ago . hi heavili armour train pull into the station just after 8am , after a 65-hour , 4,500km journey from pyongyang via mainland china . just as they did dure kim il-sung ' s visit 55 year ago , a dozen vietnames soldier in white and green uniform salut the 35-year-old leader as he emerg from the train and walk on the red carpet .

Lemmatized Kim arrive on Tuesday morning at the border station of Dong Dang and be welcome with a military salute reminiscent of the fanfare that greet his grandfather Kim Il-sung more than half a century ago . His heavily armour train pull into the station just after 8am , after a 65-hour , 4,500km journey from Pyongyang via mainland China . Just a they do during Kim Il-sung ' s visit 55 year ago , a dozen Vietnamese soldier in white and green uniform salute the 35-year-old leader a he emerge from the train and walk on the red carpet .

Sentence Segmentation

!, ? mostly unambiguous but **period** "." is very ambiguous

- Sentence boundary
- Abbreviations like Inc. or Dr.
- Numbers like .02% or 4.3. (which also can be 0,02% in many languages making the comma "," also ambiguous.

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

- An abbreviation dictionary can help

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

Word/Sentence Comparison

Comparison Metrics

How similar are two strings?

- Spell correction

  The user typed “graffe”

  Which is closest?

  - graf
  - graft
  - grail
  - giraffe

How similar are two strings?

Computational Biology

Align two sequences of nucleotides

AGGCTATCACCTGACCTCCAGGCCGATGCCC

TAGCTATCACGACCGCGGTCGATTTGCCCGAC

Resulting alignment:

-AGGCTATCACCTGACCTCCAGGCCGA--TGCCC---

TAG-CTATCAC--GACCGC--GGTCGATTTGCCCGAC

Comparing Words (Levenshtein)

The Levenshtein distance is a number that tells you how different two strings are. It measure the number of edit operations (insertion, deletion, substitution) needed to convert from string1 to string2.

The higher the number, the more different the two strings are.

$$\operatorname{lev}_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0, \\ \min \begin{cases} \operatorname{lev}_{a,b}(i-1,j) + 1 \\ \operatorname{lev}_{a,b}(i,j-1) + 1 \\ \operatorname{lev}_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases}$$

Comparing Words (Levenshtein)

Let’s see an example that there is String A: “kitten” which need to be converted in String B: “sitting” so we need to determine the minimum operation required

kitten → sitten (substitution of “s” for “k”)
sitten → sittin (substitution of “i” for “e”)
sittin → sitting (insertion of “g” at the end).

In this case it took three operation do this, so the levenshtein distance will be 3.

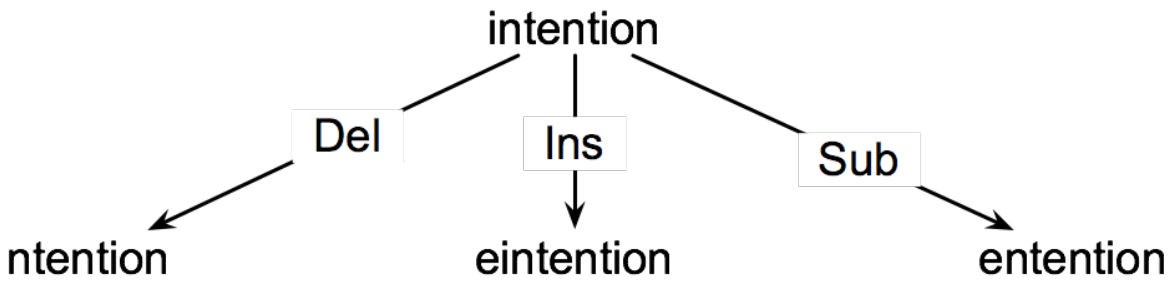
Finding the minimum distance

Searching for a path (sequence of edits) from the start string to the final string:

- Initial state: the word we're transforming
- Operators: insert, delete, substitute

Goal state: the word we're trying to get to

Path cost: what we want to minimize: the number of edits



Solution Space is VERY large.

We can’t afford to navigate naïvely

Lots of distinct paths wind up at the same state.

- We don’t have to keep track of all of them
- Just the shortest path to each of those revisted states.

Dynamic Programming Approach

- **Dynamic programming**: A tabular computation of D(*n,m*)
- Solving problems by combining solutions to subproblems.
- Bottom-up

  We compute D(i,j) for small *i,j*

  And compute larger D(i,j) based on previously computed smaller values

  i.e., compute D(*i,j*) for all *i* (0 < *i* < n) and *j* (0 < j < m)

- Initialization

```
D(i,0) = i
D(0,j) = j
```

- Recurrence Relation*:*

```
For each  i = 1…M
   For each  j = 1…N
                          D(i-1,j) + 1
              D(i,j)= min  D(i,j-1) + 1
                          D(i-1,j-1) +   2; if X(i) ≠ Y(j)
                                         0; if X(i) = Y(j)
```

- Termination*:*

```
D(N,M) is distance
```

The Edit Distance Table

N	9									
O	8									
I	7									
T	6									
N	5									
E	4									
T	3									
N	2									
I	1									
#	**0**	1	2	3	4	5	6	7	8	9
	#	E	X	E	C	U	T	I	O	N

The Edit Distance Table

$$D(i,j) = \min \begin{cases} D(i-1,j) + 1 \\ D(i,j-1) + 1 \\ D(i-1,j-1) + \begin{cases} 2; & \text{if } S_1(i) \neq S_2(j) \\ 0; & \text{if } S_1(i) = S_2(j) \end{cases} \end{cases}$$

N	9									
O	8									
I	7									
T	6									
N	5									
E	4									
T	3									
N	2									
I	1									
#	**0**	1	2	3	4	5	6	7	8	9
	#	E	X	E	C	U	T	I	O	N

Dan Jurafsky

Edit Distance

$$D(i,j) = \min \begin{cases} D(i-1,j) + 1 \\ D(i,j-1) + 1 \\ D(i-1,j-1) + \begin{cases} 2; & \text{if } S_1(i) \neq S_2(j) \\ 0; & \text{if } S_1(i) = S_2(j) \end{cases} \end{cases}$$

N	9									
O	8									
I	7									
T	6									
N	5									
E	4									
T	3									
N	2									
I	1									
#	**0**	1	2	3	4	5	6	7	8	9
	#	E	X	E	C	U	T	I	O	N

The Edit Distance Table

N	9	8	9	10	11	12	11	10	9	**8**
O	8	7	8	9	10	11	10	9	8	9
I	7	6	7	8	9	10	9	8	9	10
T	6	5	6	7	8	9	8	9	10	11
N	5	4	5	6	7	8	9	10	11	10
E	4	3	4	5	6	7	8	9	10	9
T	3	4	5	6	7	8	7	8	9	8
N	2	3	4	5	6	7	8	7	8	7
I	1	2	3	4	5	6	7	6	7	8
#	**0**	1	2	3	4	5	6	7	8	9
	#	E	X	E	C	U	T	I	O	N

For a fast C/C++ implementation

https://github.com/re-lsearch/re-lsearch/blob/master/src/fuzzy.cxx

Levenshtein Variants

Damerau-Levenshtein distance: It is similar to the Levenshtein distance, but it just also allows transpositions as an additional operation making it 4 operations.

Hamming distance: It can only be applied to strings of equal length, it is used measures the number of positions at which the corresponding characters are different.

```
int RatcliffCompare(const STRING &s1, const STRING& s2, int
scale)


 // Performs a 'fuzzy' comparison between two strings.
 Returns how
 // 'alike' they are expressed as a percentage match.
 //
 // Written originally by John W. Ratcliff for Dr. Dobbs
 Journal
 // of Software Tools Volume 13, 7, July 1988
 //
 // Pages 46, 47, 59-51, 68-72
 // http://www.ddj.com/184407970?pgno=5
 //
```

Pattern Matching: the Gestalt Approach

By John W. Ratclif, July 01, 1988

Post a Comment

String comparison routines are often limited to finding exact matches. John describes an algorithm (implemented in assembly language) that gives matches as percentages.

The best way to describe the Ratcliff/Obershelp pattern-matching algorithm, in using conventional computer terminology, is as a wild-card search that doesn't require wild cards. Instead, the algorithm creates its own wildcards, based on the closest matches found between the strings. Specifically, the algorithm works by examining two strings passed to it and locating the largest group of characters in common. The algorithm uses this group of characters as an anchor between the two strings. The algorithm then places any group of characters found to the left or the right of this anchor on a stack for further examination. This procedure is repeated for all substrings on the stack until there is nothing left to examine. The algorithm calculates the score returned as twice the number of characters found in common divided by the total number of characters in the two strings; the score is returned as an integer, reflecting a percentage match

```
int RatcliffCompare(const STRING &s1, const STRING& s2,
int scale)
{
  size_t  l1 = s1.length();
  size_t  l2 = s2.length();
  const char *p1 = s1.c_str();
  const char *p2 = s2.c_str();

  int distance = LevenshteinDistance (s1, s2);
  if (distance == 0) return scale;

  int metric =  GCsubstr(s1, p1+l1, s2, p2+l2);

  return(2 * scale * metric/ (l1+l2));
}
```

```
static int GCsubstr(const char *st1,const char *end1,const char *st2,const char *end2)
{
  const char *s1 = NULL;
  const char *s2 = NULL;
  int max = 0;
  const char *b1 = end1;
  const char *b2 = end2;
  int i;
        const char *a1;
        const char *a2;


  if (end1 <= st1) return 0;
  if (end2 <= st2) return 0;
  if (end1 == (st1+1) && end2 == (st2+1) ) return 0;

  for (a1 = st1; a1 < b1; a1++)
  {
    for (a2 = st2; a2 < b2; a2++)
    {
      if (*a1 == *a2)
                        {

                                for (i=1; a1[i] && (a1[i] == a2[i]); i++);

        if (i > max)
                                {
                                        max = i; s1 = a1; s2 = a2;
                                        b1 = end1 - max; b2 = end2 - max;
                                }

                        }
    }
  }

  if (!max) return 0;

  max += GCsubstr(s1+max, end1, s2+max, end2);
  max += GCsubstr(st1, s1, st2, s2);

  return max;
}
```

Tokenisation in Production

NLTK Tokenisers

https://www.nltk.org/api/nltk.tokenize.html

Researchers usually prefer to use NLTK because it has a variety of algorithms and with that algorithms, some tasks are very easy to perform.

Treebank

A treebank is a parsed text corpus used in linguistics. It basically has text annotated by their semantic and syntactic structure.

The English Penn Treebank (PTB) corpus, and in particular the section of the corpus corresponding to the articles of Wall Street Journal (WSJ), is one of the most known and used corpus for the evaluation of models for sequence labelling. The task consists of annotating each word with its Part-of-Speech tag. In the most common split of this corpus, sections from 0 to 18 are used for training (38 219 sentences, 912 344 tokens), sections from 19 to 21 are used for validation (5 527 sentences, 131 768 tokens), and sections from 22 to 24 are used for testing (5 462 sentences, 129 654 tokens). The corpus is also commonly used for character-level and word-level Language Modelling.

Treebank Word Tokenizer

The NTLK Treebank tokenizer uses regular expressions to tokenize text as in Penn Treebank.

This tokenizer performs the following steps:

1) split standard contractions, e.g. `don't` -> `do n't` and `they'll` -> `they 'll`
2) treat most punctuation characters as separate tokens
3) split off commas and single quotes, when followed by whitespace
4) separate periods that appear at the end of line

Treebank Regexp

```
STARTING_QUOTES = [(re.compile('^\\"'), '``'), (re.compile('(``)'), ' \\1 '),
(re.compile('([ \\(\\[{<])(\\"|\\\'{2})'), '\\1 `` ‘)]

PUNCTUATION = [(re.compile('([:,])([^\\d])'), ' \\1 \\2'), (re.compile('([:,])$'), ' \
\1 '), (re.compile('\\.\\.\\.'), ' ... '), (re.compile('[;@#$%&]'), ' \\g<0> '),
(re.compile('([^\\.])(\\.)([\\]\\)}>"\\\']*)\\s*$'), '\\1 \\2\\3 '), (re.compile('[?!]'), ' \
\g<0> '), (re.compile("([^'])' "), "\\1 ' ‘)]

PARENS_BRACKETS = (re.compile('[\\]\\[\\(\\)\\{\\}\\<\\>]'), ' \\g<0> ')

CONVERT_PARENTHESES = [(re.compile('\\('), '-LRB-'), (re.compile('\\)'), '-
RRB-'), (re.compile('\\['), '-LSB-'), (re.compile('\\]'), '-RSB-'), (re.compile('\\
{'), '-LCB-'), (re.compile('\\}'), '-RCB-')]

DOUBLE_DASHES = (re.compile('--'), ' -- ')
```

Treebank Regexp (continued)

```
ENDING_QUOTES = [(re.compile("''"), " '' "), (re.compile('"'), " '' "),
(re.compile("([^' ])('[sS]|'[mM]|'[dD]|') "), '\\1 \\2 '), (re.compile("([^' ])
('ll|'LL|'re|'RE|'ve|'VE|n't|N'T) "), '\\1 \\2 ')]

CONTRACTIONS2 = [re.compile('(?i)\\b(can)(?#X)(not)\\b',
re.IGNORECASE), re.compile("(?i)\\b(d)(?#X)('ye)\\b", re.IGNORECASE),
re.compile('(?i)\\b(gim)(?#X)(me)\\b', re.IGNORECASE), re.compile('(?i)\
\b(gon)(?#X)(na)\\b', re.IGNORECASE), re.compile('(?i)\\b(got)(?#X)(ta)\\b',
re.IGNORECASE), re.compile('(?i)\\b(lem)(?#X)(me)\\b', re.IGNORECASE),
re.compile("(?i)\\b(more)(?#X)('n)\\b", re.IGNORECASE), re.compile('(?i)\
\b(wan)(?#X)(na)(?=\\s)', re.IGNORECASE)]

CONTRACTIONS3 = [re.compile("(?i) ('t)(?#X)(is)\\b", re.IGNORECASE),
re.compile("(?i) ('t)(?#X)(was)\\b", re.IGNORECASE)]
```

Do lab02!

Summary/Discussion.

What we have noticed is that these tokenisers are perhaps not ideal for German. The algorithms in NTLK and SpaCy are great for English but …

The current “state of the art” for German is: SoMaJo

https://github.com/tsproisl/SoMaJo

Tokenizers

There is a whole lot more …

(We will return to this subject after this intro module)

Stop-Words

Wikipedia: https://en.wikipedia.org/wiki/Stop_word

“**Stop words** are the words in a **stop list** (or ***stoplist*** or *negative dictionary*) which are filtered out (i.e. stopped) before or after processing of natural language data (text) because they are deemed insignificant.”

Stop-Words

Common English stop words:

```
“A",
"about",
"above",
"according",
"across",
"actually",
"adj",
"after",
“afterwards",
..
 yours",
"yourself",
"yourselves",
"z
```

So the idea is: words like “is”, “to”, “be”, “not” are just insignificant.

Stop-Words

To these language specific stop word lists are typically also added words that have a frequency over a specific threshold.

A search engine for steel parts where every record contains the word “steel” would seem a natural stop word, right?

Some software algorithms such as the popular inverted index algorithm used by Lucene (the engine behind Elasticsearch, Neo4j and others) for word/term search (full-text) needs stop words to keep their tables from exploiding— they also need to limit term length, number of fields etc. (but that’s another talk)— despite their use of TD-IDF and BM25 algorithms do down rank frequent terms.

Typical workflow

1) Tokenise.
2) [Normalize] / [Correct]
3) [Stem/Lemmatization]

Stemming is often used by search engines, information retrieval, and text mining. Lemmatization is essential for chatbots, text classification, and semantic analysis.

4) [Remove stop words]

Stop-Words. Why not use them?

- “To be or not to be” / “Sein oder nicht sein” — all stop words but as one of the most famous lines from Shakespeare
“To be, or not to be: that is the question:"
clearly NOT without meaning.

- Removing stop words can lead to poor recall as sometimes they are important to understand the word itself.

- Space is no longer the great premium. In 1956, an IBM 5 MB hard disk drive cost $50K and weighed over 1000 Kg.
When the IBM PC was launched in 1981 one could order a 5 MB HD for around $3000. It was expensive. In 1992 the first 1 GB hard disk was launched by DEC. Today we can buy a 8 TB HDD for as little as $150 USD and excellent 8TB SDDs (flash file storage) cost under $600 USD.

Stop-Words. Why not use them?

In my own search-engine I support the use of stop words BUT strongly discourage their use. I use different algorithms from the inverted index so don’t need them (and place no limits on term length, number of paths/fields etc.).

With embeddings (in search augmented LLM) stop words is not uncommon but I’d argue the wrong conclusion to a problem.

Word2Vec, a common algorithm down samples frequent words.

B. Probability Refresher

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: $$\mathrm{P}(\bigcup_{i}^{\infty} A_i) = \sum_{i}^{\infty} \mathrm{P}(A_i)$$

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

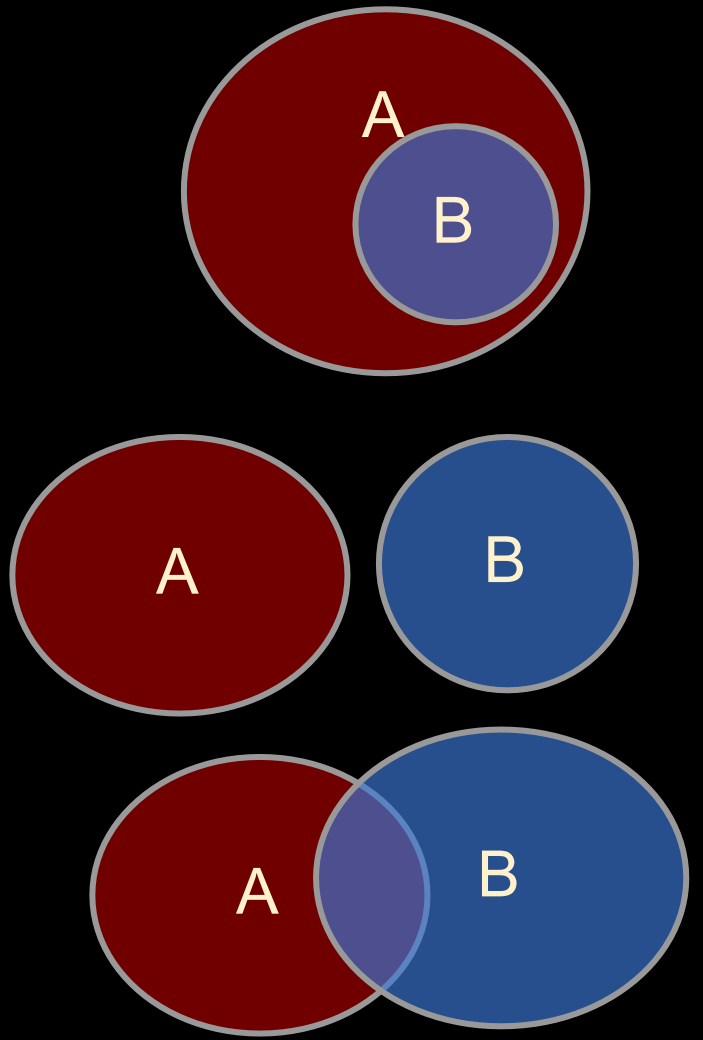
P is a *probability measure*, if and only if

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

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

Probability

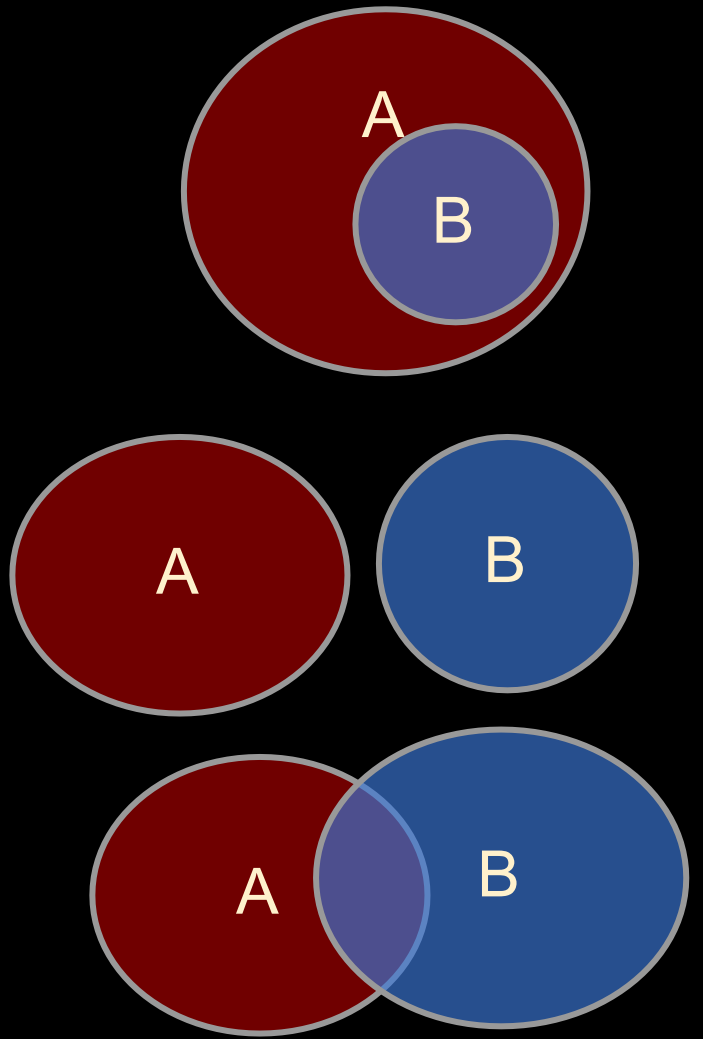
Some Properties:



Probability

Some Properties:

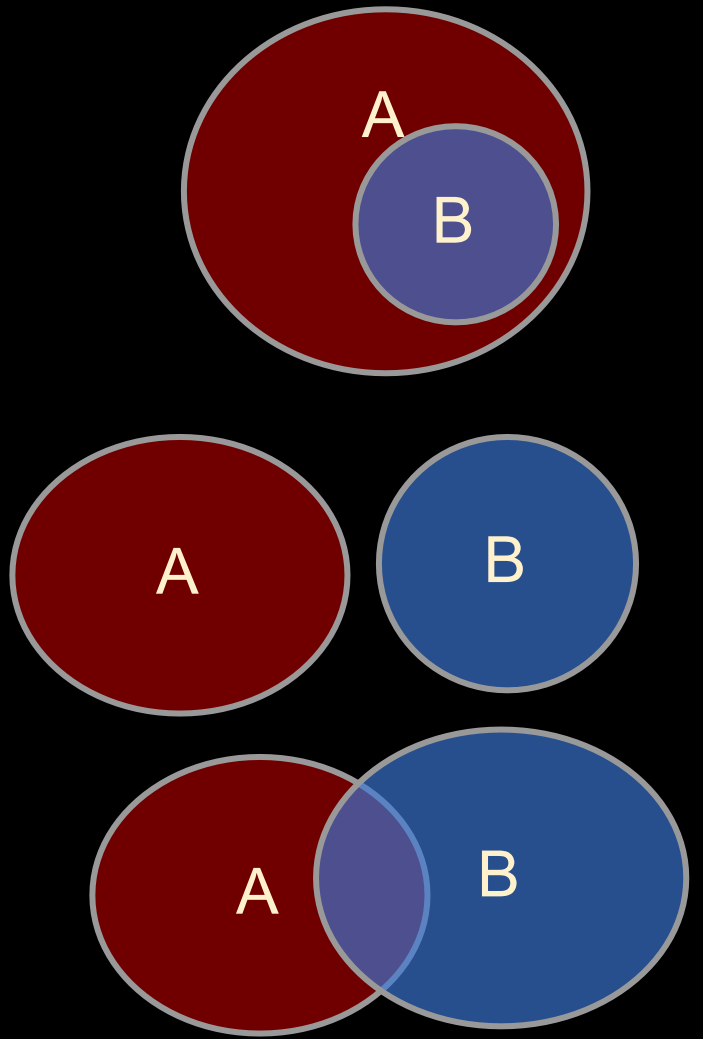
1. If $B \subseteq A$ then $P(A) \geq P(B)$



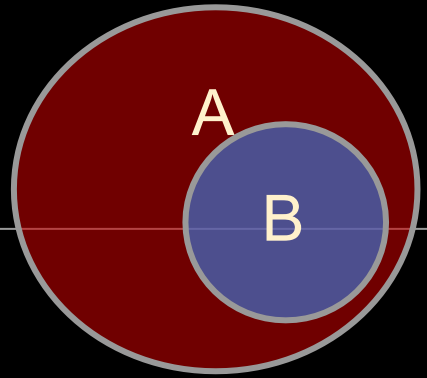
Probability

Some Properties:

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



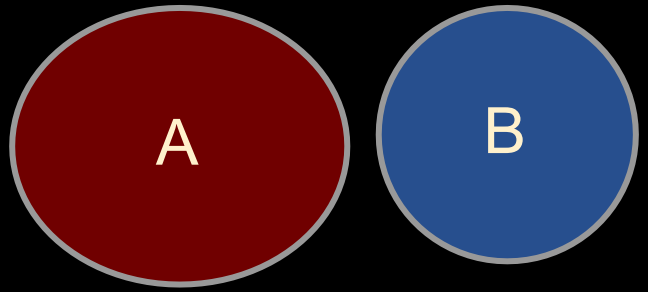
Probability

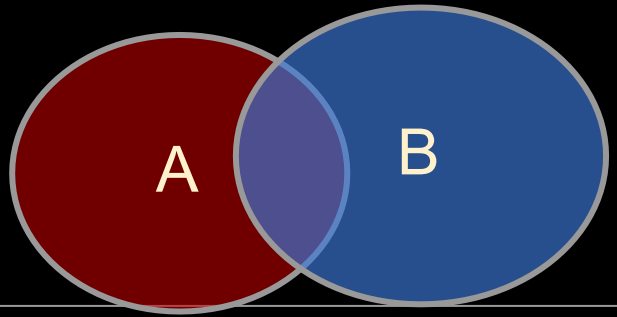


Some Properties:

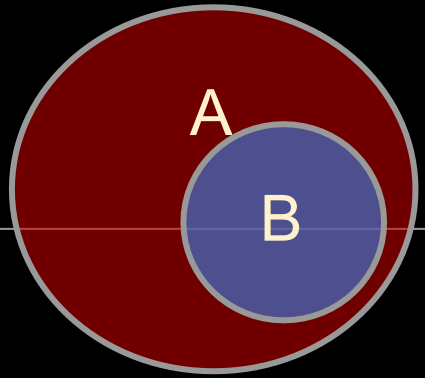
1. If $B \subseteq A$ then $P(A) \geq P(B)$
2. $P(A \cup B) \leq P(A) + P(B)$
3. $P(A \cap B) \leq \min(P(A), P(B))$
4. $P(\neg A) = P(\Omega / A) = 1 - P(A)$

/ is set difference





Probability

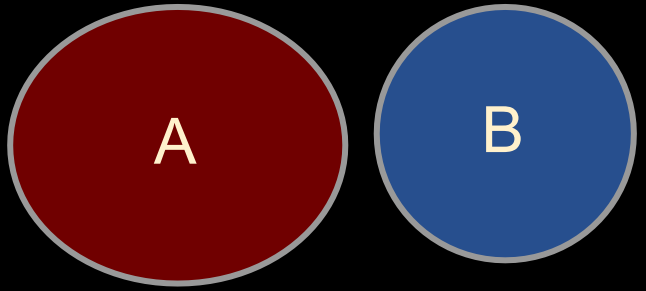


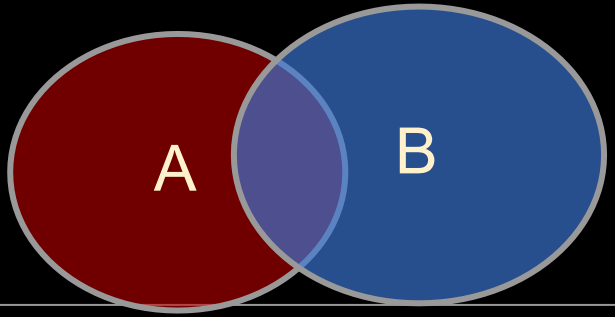
Some Properties:

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: $P(A, B) = P(A)P(B)$

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

Chain Rule

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

Rewriting: $P(A,B) = P(A)\,P(B|A)$

For A,B,C,D

$$P(A,B,C,D) = P(A)\,P(B|A)\,P(C|A,B)\,P(D|A,B,C)$$

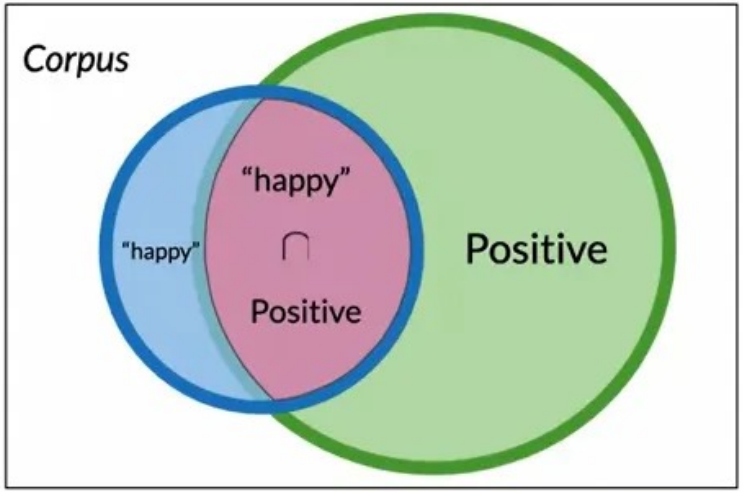
Bayes Theorem (Law)

$$P(A|B) = P(B|A) \times P(A) / P(B)$$

Bayesian philosophy : degrees of belief

Example (Sentiment)

Conditional probabilities



$$P(\text{Positive}|\text{“happy”}) = \frac{P(\text{Positive} \cap \text{“happy”})}{P(\text{“happy”})}$$

Conditionals

$$P(\text{Positive}|\text{“happy”}) = \frac{P(\text{Positive} \cap \text{“happy”})}{P(\text{“happy”})}$$

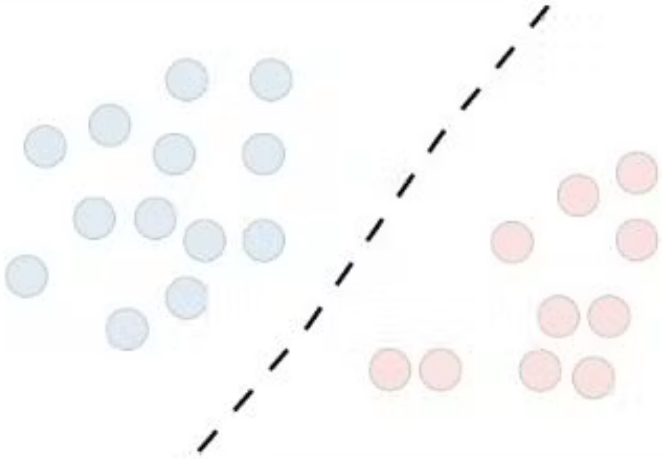
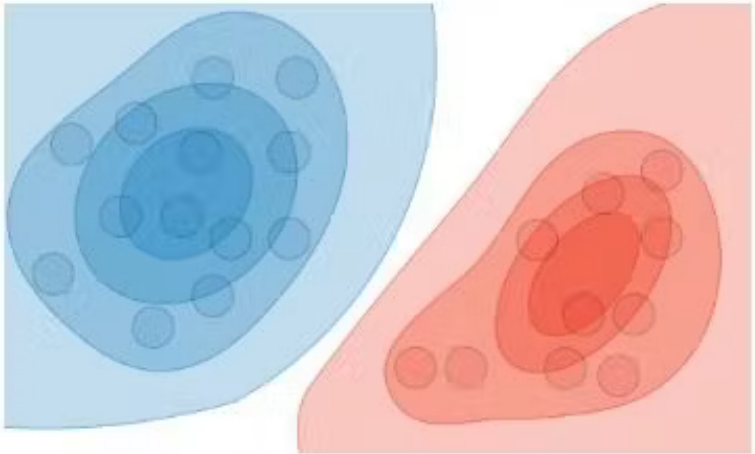
$$P(\text{“happy”}|\text{Positive}) = \frac{P(\text{“happy”} \cap \text{Positive})}{P(\text{Positive})}$$

Conditionals

$$P(\text{Positive}|\text{“happy”}) = P(\text{“happy”}|\text{Positive}) \times \frac{P(\text{Positive})}{P(\text{“happy”})}$$

$$\boxed{P(X|Y) = P(Y|X) \times \frac{P(X)}{P(Y)}}$$

Generative versus Discrimative

	Discriminative model	Generative model
Goal	Directly estimate $P(y\|x)$	Estimate $P(x\|y)$ to then deduce $P(y\|x)$
What's learned	Decision boundary	Probability distributions of the data
Illustration		
Examples	Regressions, SVMs	GDA, Naive Bayes

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)

C. Classifiers

Text Classification and Naive Bases

The Task of Text Classification

Use Classifier slide Stack…

Returned from the other slides

D. N-Grams

N-Grams (Wikipedia)

An ***n*-gram** is a sequence of *n* adjacent symbols in particular order. The symbols may be *n* adjacent letters (including punctuation marks and blanks), syllables, or rarely whole words found in a language dataset; or adjacent phonemes extracted from a speech-recording dataset, or adjacent base pairs extracted from a genome. They are collected from a text corpus or speech corpus.

Predicting words

The water of Walden Pond is beautifully ...

blue
green
clear

*refrigerator
*that

Language Models

Systems that can predict upcoming words

- Can assign a probability to each potential next word
- Can assign a probability to a whole sentence

Why word prediction?

It's a helpful part of language tasks

- Grammar or spell checking

  Their are two midterms ~~Their~~ There are two midterms

  Everything has improve Everything has ~~improve~~ improved

- Speech recognition

  I will be back soonish I will be bassoon dish

Why word prediction?

It's how **large language models (LLMs)** work!

LLMs are **trained** to predict words

- Left-to-right (autoregressive) LMs learn to predict next word

LLMs **generate** text by predicting words

- By predicting the next word over and over again

Language Modeling (LM) more formally

Goal: compute the probability of a sentence or sequence of words W:

$$P(W) = P(w_1,w_2,w_3,w_4,w_5 \ldots w_n)$$

Related task: probability of an upcoming word:

$$P(w_5|w_1,w_2,w_3,w_4) \text{ or } P(w_n|w_1,w_2 \ldots w_{n-1})$$

An LM computes either of these:

$$P(W) \quad \text{or} \quad P(w_n|w_1,w_2 \ldots w_{n-1})$$

How to estimate these probabilities

Could we just count and divide?

$$P(\text{blue}|\text{The water of Walden Pond is so beautifully}) = \frac{C(\text{The water of Walden Pond is so beautifully blue})}{C(\text{The water of Walden Pond is so beautifully})}$$

No! Too many possible sentences!

We'll never see enough data for estimating these

How to compute P(W) or $P(w_n | w_1, \ldots w_{n-1})$

How to compute the joint probability P(W):

P(The, water, of, Walden, Pond, is, so, beautifully, blue)

Intuition: let's rely on the Chain Rule of Probability

Reminder: The Chain Rule

Recall the definition of conditional probabilities

$$P(B|A) = P(A,B)/P(A) \qquad \text{Rewriting:} \quad P(A,B) = P(A)\,P(B|A)$$

More variables:

$$P(A,B,C,D) = P(A)\,P(B|A)\,P(C|A,B)\,P(D|A,B,C)$$

The Chain Rule in General

$$P(x_1,x_2,x_3,\ldots,x_n) = P(x_1)P(x_2|x_1)P(x_3|x_1,x_2)\ldots P(x_n|x_1,\ldots,x_{n-1})$$

The Chain Rule applied to compute joint probability of words in sentence

$$P(w_{1:n}) = P(w_1)P(w_2|w_1)P(w_3|w_{1:2})\ldots P(w_n|w_{1:n-1})$$
$$= \prod_{k=1}^{n} P(w_k|w_{1:k-1})$$

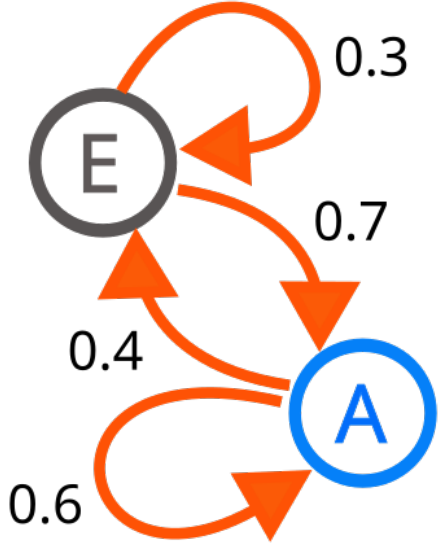
P("The water of Walden Pond") =

P(The) × P(water|The) × P(of|The water)

× P(Walden|The water of) × P(Pond|The water of Walden

Markov chain

A **Markov chain** or **Markov process** is a stochastic process describing a sequence of possible events in which the probability of each event depends only on the state attained in the previous event.



Markov Assumption

Simplifying assumption:

$$P(\text{blue}|\text{The water of Walden Pond is so beautifully})$$

$$\approx P(\text{blue}|\text{beautifully})$$

$$P(w_n|w_{1:n-1}) \approx P(w_n|w_{n-1})$$

Wikimedia commons

Bigram Markov Assumption

$$P(w_{1:n}) \approx \prod_{k=1}^{n} P(w_k|w_{k-1})$$

Instead of: $\prod_{k=1}^{n} P(w_k|w_{1:k-1})$

More generally, we approximate each component in the product

$$P(w_n|w_{1:n-1}) \approx P(w_n|w_{n-N+1:n-1})$$

Simplest case: Unigram (1-gram) model

Some automatically generated sentences from two different unigram models

```
To him swallowed confess hear both . Which . Of save on trail for
are ay device and rote life have

Hill he late speaks ; or ! a more to leg less first you enter

Months the my and issue of year foreign new exchange's September

were recession exchange new endorsed a acquire to six executives
```

Bigram (2-gram) model

Some automatically generated sentences rom two different unigram models

```
Why dost stand forth thy canopy, forsooth; he is this palpable hit
the King Henry. Live king. Follow.

What means, sir. I confess she? then all sorts, he is trim, captain.

Last December through the way to preserve the Hudson corporation N.
B. E. C. Taylor would seem to complete the major central planners one
gram point five percent of U. S. E. has already old M. X. corporation
of living

on information such as more frequently fishing to keep her
```

Problems with N-gram models

- N-grams can't handle **long-distance dependencies**:

"**The soups** that I made from that new cookbook I bought yesterday **were** amazingly delicious."

- N-grams don't do well at modeling new sequences with similar meanings

The solution: **Large language models**

- can handle much longer contexts
- because of using embedding spaces, can model synonymy better, and generate better novel strings

Why N-gram models?

A nice clear paradigm that lets us introduce many of the important issues for **large language models**

- **training** and **test** sets
- the **perplexity** metric
- **sampling** to generate sentences
- ideas like **interpolation** and **backoff**

Estimating bigram probabilities

The Maximum Likelihood Estimate

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{\sum_w C(w_{n-1}w)}$$

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})}$$

An example

<s> I am Sam </s>

<s> Sam I am </s>

<s> I do not like green eggs and ham </s>

$$P(\texttt{I}\,|\,\texttt{<s>}) = \frac{2}{3} = .67 \qquad P(\texttt{Sam}\,|\,\texttt{<s>}) = \frac{1}{3} = .33 \qquad P(\texttt{am}\,|\,\texttt{I}) = \frac{2}{3} = .67$$

$$P(\texttt{</s>}\,|\,\texttt{Sam}) = \frac{1}{2} = 0.5 \qquad P(\texttt{Sam}\,|\,\texttt{am}) = \frac{1}{2} = .5 \qquad P(\texttt{do}\,|\,\texttt{I}) = \frac{1}{3} = .33$$

More examples:
Berkeley Restaurant Project sentences

can you tell me about any good cantonese restaurants close by

tell me about chez panisse

i’m looking for a good place to eat breakfast

when is caffe venezia open during the day

Estimating bigram probabilities

The Maximum Likelihood Estimate

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{\sum_w C(w_{n-1}w)}$$

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})}$$

An example

<s> I am Sam </s>

<s> Sam I am </s>

<s> I do not like green eggs and ham </s>

$$P(\texttt{I}\,|\,\texttt{<s>}) = \frac{2}{3} = .67 \qquad P(\texttt{Sam}\,|\,\texttt{<s>}) = \frac{1}{3} = .33 \qquad P(\texttt{am}\,|\,\texttt{I}) = \frac{2}{3} = .67$$

$$P(\texttt{</s>}\,|\,\texttt{Sam}) = \frac{1}{2} = 0.5 \qquad P(\texttt{Sam}\,|\,\texttt{am}) = \frac{1}{2} = .5 \qquad P(\texttt{do}\,|\,\texttt{I}) = \frac{1}{3} = .33$$

More examples:
Berkeley Restaurant Project sentences

can you tell me about any good cantonese restaurants close by

tell me about chez panisse

i’m looking for a good place to eat breakfast

when is caffe venezia open during the day

Raw bigram counts

Out of 9222 sentences

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

Raw bigram probabilities

Normalize by unigrams:

Result:

i	want	to	eat	chinese	food	lunch	spend
2533	927	2417	746	158	1093	341	278

	i	want	to	eat	chinese	food	lunch	spend
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

Bigram estimates of sentence probabilities

$$P(\langle s\rangle \text{ I want english food } \langle /s\rangle) =$$
$$P(\text{I}|\langle s\rangle)$$
$$\times\ P(\text{want}|\text{I})$$
$$\times\ P(\text{english}|\text{want})$$
$$\times\ P(\text{food}|\text{english})$$
$$\times\ P(\langle /s\rangle|\text{food})$$
$$=\ .000031$$

What kinds of knowledge do N-grams represent?

P(english|want) = .0011
P(chinese|want) = .0065
P(to|want) = .66
P(eat | to) = .28
P(food | to) = 0
P(want | spend) = 0
P (i | <s>) = .25

Dealing with scale in large n-grams

LM probabilities are stored and computed in log format, i.e. **log probabilities**

This avoids underflow from multiplying many small numbers

If we need probabilities we can do one exp at the end

$$p_1 \times p_2 \times p_3 \times p_4 = \exp(\log p_1 + \log p_2 + \log p_3 + \log p_4)$$

Larger ngrams

4-grams, 5-grams

Large datasets of large n-grams have been released

- N-grams from Corpus of Contemporary American English (COCA) 1 billion words (Davies 2020)
- Google Web 5-grams (Franz and Brants 2006) 1 trillion words)
- Efficiency: quantize probabilities to 4-8 bits instead of 8-byte float

Newest model: infini-grams (∞-grams) (Liu et al 2024)

- No precomputing! Instead, store 5 trillion words of web text in **suffix arrays**. Can compute n-gram probabilities with any n!

N-gram LM Toolkits

SRILM

- http://www.speech.sri.com/projects/srilm/

KenLM

- https://kheafield.com/code/kenlm/

More on Sentiment Classification

Relationship to Language Modelling

Text Classification and Naïve Bayes

Naïve Bayes: Relationship to Language Modeling

Evaluating Classifiers: How well does our classifier work?

Let's first address binary classifiers:

- Is this email spam?

  spam (+) or not spam (-)

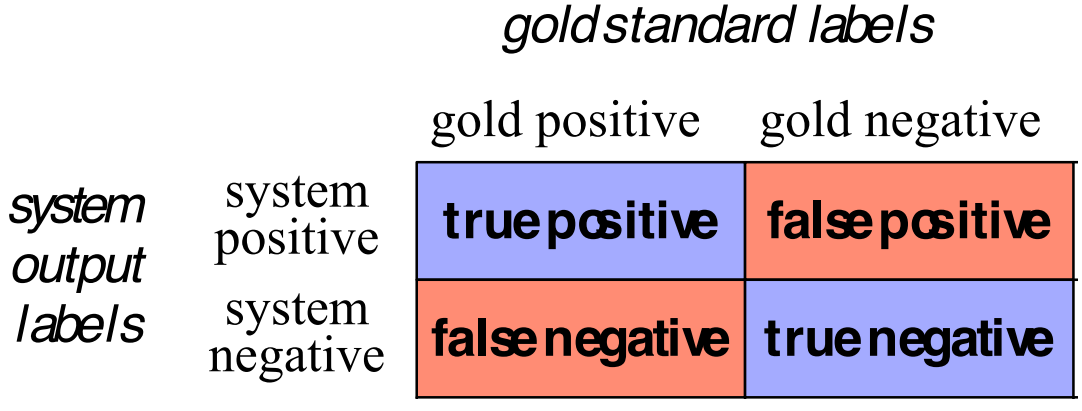
- Is this post about Delicious Pie Company?

  about Del. Pie Co (+) or not about Del. Pie Co(-)

We'll need to know

1. What did our classifier say about each email or post?
2. What should our classifier have said, i.e., the correct answer, usually as defined by humans ("gold label")

First step in evaluation: The confusion matrix



Accuracy on the confusion matrix

gold standard labels

system output labels		gold positive	gold negative
	system positive	**true positive**	**false positive**
	system negative	**false negative**	**true negative**

$$\textbf{accuracy} = \frac{tp+tn}{tp+fp+tn+fn}$$

Why don't we use accuracy?

Accuracy doesn't work well when we're dealing with uncommon or imbalanced classes

Suppose we look at 1,000,000 social media posts to find Delicious Pie-lovers (or haters)

- 100 of them talk about our pie
- 999,900 are posts about something unrelated

Imagine the following simple classifier

Every post is "not about pie"

Accuracy re: pie posts

100 posts are about pie; 999,900 are

gold standard labels

system output labels	gold positive	gold negative
system positive	**true positive**	**false positive**
system negative	**false negative**	**true negative**

$$\textbf{accuracy} = \frac{tp+tn}{tp+fp+tn+fn}$$

Why don't we use accuracy?

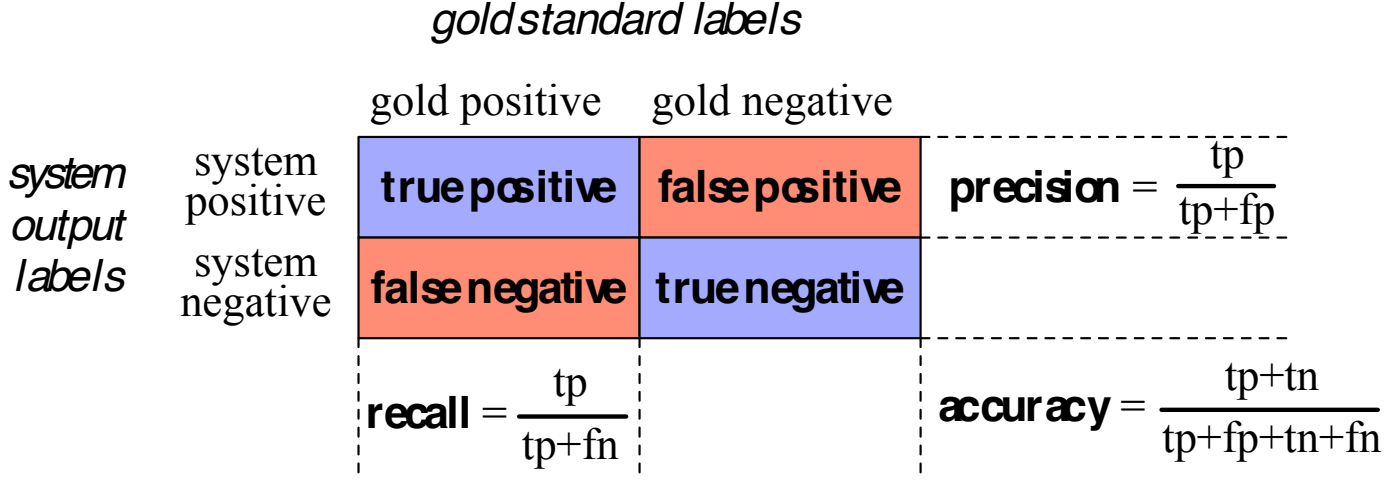
Accuracy of our "nothing is pie" classifier

- 999,900 true negatives and 100 false negatives
- Accuracy is 999,900/1,000,000 = 99.99%!
- But useless at finding pie-lovers (or haters)!!
- Which was our goal!

Accuracy doesn't work well for unbalanced classes

- Most tweets are not about pie!

Instead of accuracy we use precision and recall



Precision: % of selected items that are correct

Recall: % of correct items that are selected

Precision/Recall aren't fooled by the"just call everything negative" classifier!

Stupid classifier: Just say no: every tweet is "not about pie"

- 100 tweets talk about pie, 999,900 tweets don't
- Accuracy = 999,900/1,000,000 = 99.99%

But the Recall and Precision for this classifier are terrible:

$$\textbf{Recall} = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$$

$$\textbf{Precision} = \frac{\text{true positives}}{\text{true positives} + \text{false positives}}$$

A combined measure: F1

F1 is a combination of precision and recall.

$$F_1 = \frac{2PR}{P+R}$$

F1 is a special case of the general "F-measure"

F-measure is the (weighted) harmonic mean of precision and recall

$$\text{HarmonicMean}(a_1, a_2, a_3, a_4, \ldots, a_n) = \frac{n}{\frac{1}{a_1} + \frac{1}{a_2} + \frac{1}{a_3} + \ldots + \frac{1}{a_n}}$$

$$F = \frac{1}{\alpha \frac{1}{P} + (1-\alpha)\frac{1}{R}} \quad \text{or} \left(\text{with } \beta^2 = \frac{1-\alpha}{\alpha} \right) \quad F = \frac{(\beta^2 + 1)PR}{\beta^2 P + R}$$

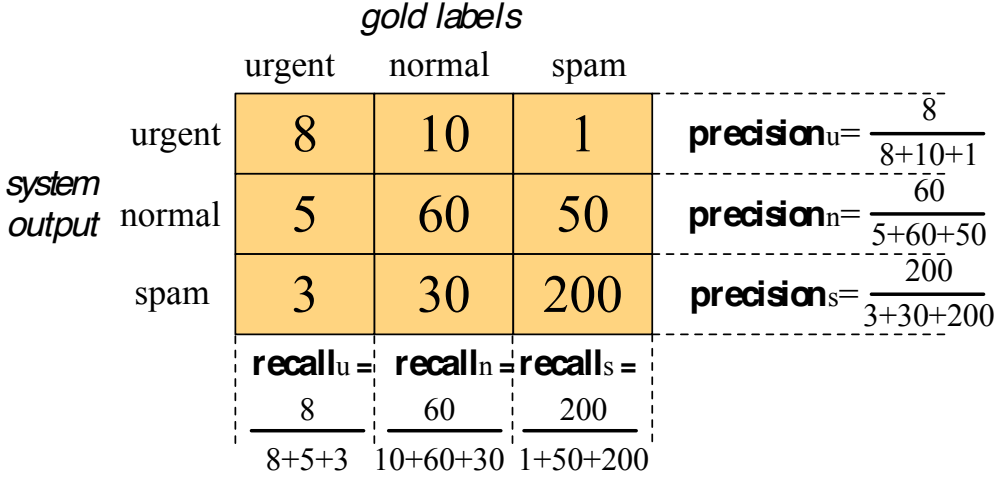
F1 is a special case of F-measure with $\beta=1$, $\alpha=½$

Suppose we have more than 2 classes?

Lots of text classification tasks have more than two classes.

- Sentiment analysis (positive, negative, neutral) , named entities (person, location, organization)

We can define precision and recall for multiple classes like this 3-way email task:



How to combine P/R values for different classes: Microaveraging vs Macroaveraging

Class 1: Urgent

	true urgent	true not
system urgent	8	11
system not	8	340

$$\text{precision} = \frac{8}{8+11} = .42$$

Class 2: Normal

	true normal	true not
system normal	60	55
system not	40	212

$$\text{precision} = \frac{60}{60+55} = .52$$

Class 3: Spam

	true spam	true not
system spam	200	33
system not	51	83

$$\text{precision} = \frac{200}{200+33} = .86$$

Pooled

	true yes	true no
system yes	268	99
system no	99	635

$$\text{microaverage precision} = \frac{268}{268+99} = \mathbf{.73}$$

$$\text{macroaverage precision} = \frac{.42+.52+.86}{3} = \mathbf{.60}$$

Text Classification and Naive Bayes

Precision, Recall, and F1

Text Classification and Naive Bayes

Avoiding Harms in Classification

Harms of classification

Classifiers, like any NLP algorithm, can cause harms

This is true for any classifier, whether Naive Bayes or other algorithms

Representational Harms

- Harms caused by a system that demeans a social group
  - Such as by perpetuating negative stereotypes about them.
- Kiritchenko and Mohammad 2018 study
  - Examined 200 sentiment analysis systems on pairs of sentences
  - Identical except for names:
    - common African American (Shaniqua) or European American (Stephanie).
    - Like "I talked to Shaniqua yesterday" vs "I talked to Stephanie yesterday"
- Result: systems assigned lower sentiment and more negative emotion to sentences with African American names
- Downstream harm:
  - Perpetuates stereotypes about African Americans
  - African Americans treated differently by NLP tools like sentiment (widely used in marketing research, mental health studies, etc.)

Harms of Censorship

- **Toxicity detection** is the text classification task of detecting hate speech, abuse, harassment, or other kinds of toxic language.
  - Widely used in online content moderation
- Toxicity classifiers incorrectly flag non-toxic sentences that simply mention minority identities (like the words "blind" or "gay")
  - women (Park et al., 2018),
  - disabled people (Hutchinson et al., 2020)
  - gay people (Dixon et al., 2018; Oliva et al., 2021)
- Downstream harms:
  - Censorship of speech by disabled people and other groups
  - Speech by these groups becomes less visible online
  - Writers might be nudged by these algorithms to avoid these words making people less likely to write about themselves or these groups.

Performance Disparities

1. Text classifiers perform worse on many **languages** of the world due to lack of data or labels
2. Text classifiers perform worse on **varieties** of even high-resource languages like English
   - Example task: language identification, a first step in NLP pipeline ("Is this post in English or not?")
   - English language detection performance worse for writers who are African American (Blodgett and O'Connor 2017) or from India (Jurgens et al., 2017)

Harms in text classification

- **Causes:**
  - Issues in the data; NLP systems amplify biases in training data
  - Problems in the labels
  - Problems in the algorithms (like what the model is trained to optimize)
- **Prevalence**: The same problems occur throughout NLP (including large language models)
- **Solutions**: There are no general mitigations or solutions
  - But harm mitigation is an active area of research
  - And there are standard benchmarks and tools that we can use for measuring some of the harms

Named Entity Recognition

- Rule Based
- Machine Learning
- Attention/Large Language Models

Named Entity Recognition and Classification

<PER>**Prof. Jerry Hobbs**</PER> taught CS544 during <DATE>**February 2010**</DATE>.
<PER>**Jerry Hobbs**</PER> killed his daughter in <LOC>**Ohio**</LOC>.
<ORG>**Hobbs corporation**</ORG> bought <ORG>**FbK**</ORG>.

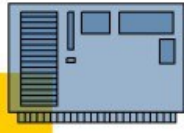
- Identify mentions in text and classify them into a predefined set of categories of interest:
  - Person Names: **Prof. Jerry Hobbs**, **Jerry Hobbs**
  - Organizations: **Hobbs corporation**, **FbK**
  - Locations: **Ohio**
  - Date and time expressions: **February 2010**
  - E-mail: **mkg@gmail.com**
  - Web address: **www.usc.edu**
  - Names of drugs: **paracetamol**
  - Names of ships: **Queen Marry**
  - Bibliographic references:
  - …

Knowledge NER vs. Learning NER

Knowledge Engineering

+ very precise (hand-coded rules)
+ small amount of training data
- expensive development & test cycle
- domain dependent
- changes over time are hard

Learning Systems



+ higher recall
+ no need to develop grammars
+ developers do not need to be experts
+ annotations are cheap
- require lots of training data

Rule Based NER (1)

- **Create regular expressions to extract:**
  - Telephone number
  - E-mail
  - Capitalized names

blocks of digits separated by hyphens

RegEx = (\d+\-)+\d+

- matches valid phone numbers like 900-865-1125 and 725-1234
- incorrectly extracts social security numbers 123-45-6789
- fails to identify numbers like 800.865.1125 and (800)865-CARE

Improved RegEx = (\d{3}[-.\ ()]){1,2}[\dA-Z]{4}

Rule Based NER (1)

- **Create regular expressions to extract:**
  - Telephone number
  - E-mail
  - Capitalized names

blocks of digits separated by hyphens

RegEx = (\d+\-)+\d+

- matches valid phone numbers like 900-865-1125 and 725-1234
- incorrectly extracts social security numbers 123-45-6789
- fails to identify numbers like 800.865.1125 and (800)865-CARE

Improved RegEx = (\d{3}[-.\ ()]){1,2}[\dA-Z]{4}

Perl RegEx

- \w (word char) any alpha-numeric
- \d (digit char) any digit
- \s (space char) any whitespace
- . (wildcard) anything
- \b word bounday
- ^ beginning of string
- $ end of string
- ? For 0 or 1 occurrences
- + for 1 or more occurrences
- specific range of number of occurrences: {min,max}.
  - A{1,5} One to five A's.
  - A{5,} Five or more A's
  - A{5} Exactly five A's

Rule Based NER (1)

- **Create regular expressions to extract:**
  - Telephone number
  - E-mail
  - Capitalized names

blocks of digits separated by hyphens

RegEx = (\d+\-)+\d+

- matches valid phone numbers like 900-865-1125 and 725-1234
- incorrectly extracts social security numbers 123-45-6789
- fails to identify numbers like 800.865.1125 and (800)865-CARE

Improved RegEx = (\d{3}[-.\ ()]){1,2}[\dA-Z]{4}

Rule Based NER (1)

- **Create regular expressions to extract:**
  - Telephone number
  - E-mail
  - Capitalized names

blocks of digits separated by hyphens

RegEx = (\d+\-)+\d+

- matches valid phone numbers like 900-865-1125 and 725-1234
- incorrectly extracts social security numbers 123-45-6789
- fails to identify numbers like 800.865.1125 and (800)865-CARE

Improved RegEx = (\d{3}[-.\ ()]){1,2}[\dA-Z]{4}

Rule Based NER (2)

- **Create rules to extract locations**
  - Capitalized word + {city, center, river} indicates location

    Ex. *New York city*

    Hudson river

  - Capitalized word + {street, boulevard, avenue} indicates location

    Ex. *Fifth avenue*

Rule Based NER (3)

- **Use context patterns**

  still not so simple:

  – [*PERSON*|*ORGANIZATION*|*ANIMAL*] fly to [*LOCATION*|*PERSON*|*EVENT*]

  Ex. *Jerry flew to Japan*

  Sarah flies to the party

  Delta flies to Europe

  bird flies to trees

  bee flies to the wood

Rule Based NER (3)

- **Use context patterns**

  still not so simple:

  – [*PERSON*|*ORGANIZATION*|*ANIMAL*] fly to [*LOCATION*|*PERSON*|*EVENT*]

  Ex. *Jerry flew to Japan*

  Sarah flies to the party

  Delta flies to Europe

  bird flies to trees

  bee flies to the wood

Why simple things would not work?

- The same entity can have multiple variants of the same proper name

  Zornitsa Kozareva

  prof. Kozareva

  Zori

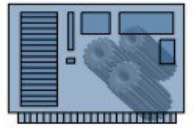
- Proper names are ambiguous

  Jordan the *person* vs. Jordan the *location*

  JFK the *person* vs. JFK the *airport*

  May the *person* vs. May the *month*

Learning System



- *Semi-supervised* learning
  - small percentage of training examples are labeled, the rest is unlabeled
  - methods: bootstrapping, active learning, co-training, self-training
  - example: NE recognition, POS tagging, Parsing, …

Machine Learning NER

Adam_B-PER Smith_I-PER works_O for_O **IBM_B-ORG** ,_O **London_B-LOC** ._O

- **NED**: Identify named entities using BIO tags
  - B beginning of an entity
  - I continues the entity
  - O word outside the entity
- **NEC**: Classify into a predefined set of categories
  - Person names
  - Organizations (companies, governmental organizations, etc.)
  - Locations (cities, countries, etc.)
  - Miscellaneous (movie titles, sport events, etc.)

Machine Learning NER

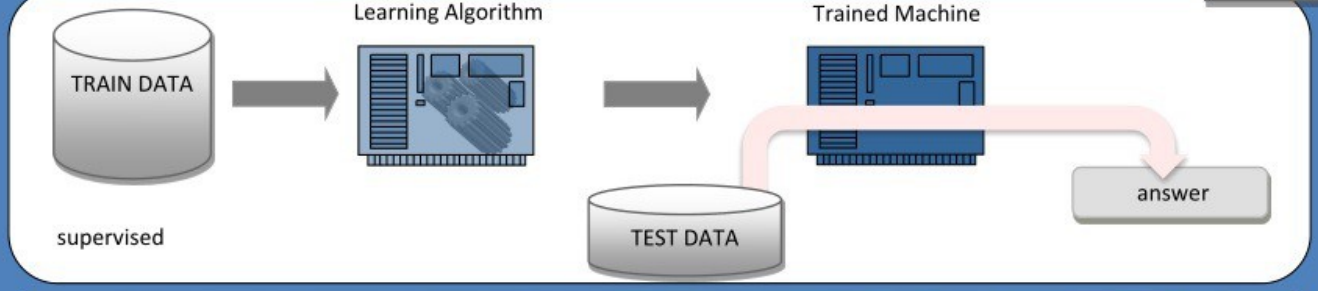
Adam_B-PER Smith_I-PER works_O for_O **IBM_B-ORG** ,_O **London_B-LOC** ._O

- **NED**: Identify named entities using BIO tags
  - B beginning of an entity
  - I continues the entity
  - O word outside the entity
- **NEC**: Classify into a predefined set of categories
  - Person names
  - Organizations (companies, governmental organizations, etc.)
  - Locations (cities, countries, etc.)
  - Miscellaneous (movie titles, sport events, etc.)

United States presidential election of 2008, scheduled for Tuesday November 4, 2008, will be the 56th consecutive quadernnial United States presidential election and will select the President and the Vice President of the United States. The Republican Party has chosen John McCain, the senior United States Senator from Arizona as its nominee; the Democratic Party has chosen Barak Obama, the junior United States Senator from Illinois, as its nominee.

Named Entity Detection

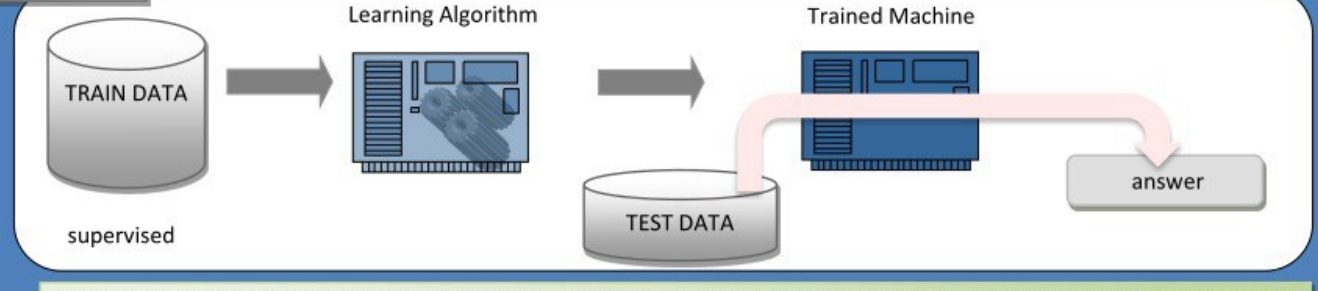
United,0,1,0,0,1,ed,ted,Un,Uni,1,0,1,1,1,null,null,null,States,presidential,election,1,1,1,0,0



United_B States_I presidential_O election_O of_O 2008_O ,_O scheduled_O for_O Tuesday_O November_O 4_O ,_O 2008_O ,_O will_O be_O the_O 56th_O consecutive_O quadernnial_O United_B States_I presidential_O election_O and_O will_O select_O the_O President_B and_O the_O Vice_B President_I of_I the_I United_I States_I. The_O Republican_B Party_I has_O chosen_O John_B McCain_I ,_O the_O senior_O United_B

Named Entity Classification

United,null,null,null,States,presidential,election,1,1,1,1,0,1,0,0,1,1,1,1,1,0,1,0,1,1,ed,ted,Uni,Un



United_B-LOC States_I-LOC presidential_O election_O of_O 2008_O ,_O scheduled_O for_O Tuesday_O November_O 4_O ,_O 2008_O ,_O will_O be_O the_O 56th_O consecutive_O quadernnial_O United_B-LOC States_I-LOC presidential_O election_O and_O will_O select_O the_O President_B-PER and_O the_O Vice_B-PER President_I-PER of_I-PER the_I-PER United_I-PER States_I-PER. The_O Republican_B-ORG Party_I-ORG has_O chosen_O John_B-PER McCain_I-PER ,_O the_O senior_O United_B-PER States_I-PER Senator_I-PER from_O Arizona_B-LOC as_O its_O

Learning for Classification

- A training example is an instance $x \in X$, paired with its correct category $c(x)$: $\langle x, c(x) \rangle$ for an unknown categorization function, c.
- Given:
  - A set of training examples, T.
  - A hypothesis space, H, of possible categorization functions, $h(x)$.
- Find a consistent hypothesis, $h(x) \in H$, such that:

$$\forall \langle x, c(x) \rangle \in T : h(x) = c(x)$$

k Nearest Neighbor

- Learning is just storing the representations of the training examples.

- Testing instance x_p:
  - compute similarity between x_p and all training examples
  - take vote among x_p k nearest neighbours
  - assign x_p with the category of the most similar example in T

Distance measures

- Nearest neighbor method uses similarity (or distance) metric.

- Given two objects *x* and *y* both with *n* values

$$x = (x_1, x_2, \ldots, x_n)$$
$$y = (y_1, y_2, \ldots, y_n)$$

calculate the Euclidean distance as

$$d(x,y) = \sqrt{\sum_{i=1}^{p} |x_i - y_i|^2}$$

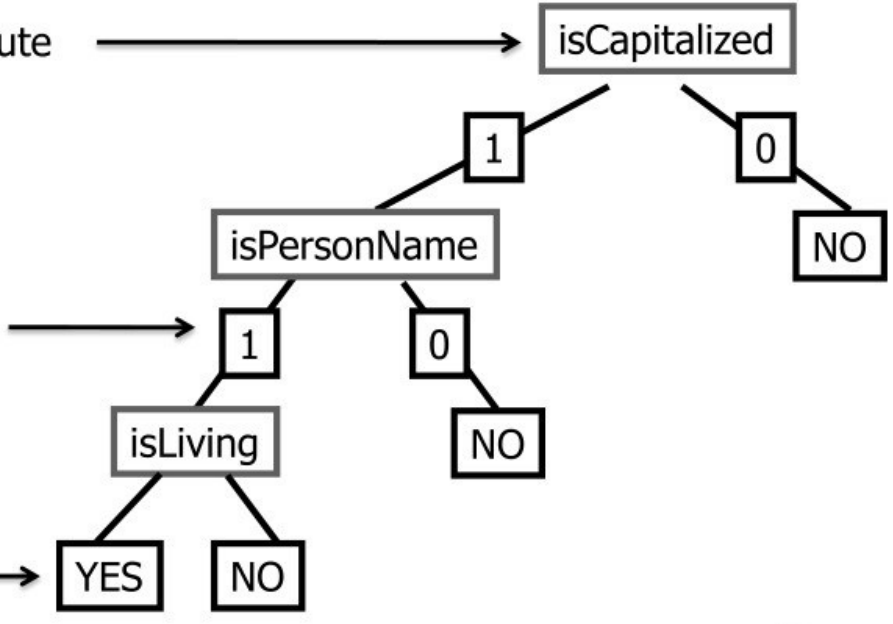
An Example

	isPersonNam e	isCapitalized	isLiving	X is PersonName?
profession	0	0	0	NO
Jerry Hobbs	1	1	1	YES
USC	0	1	0	NO
Jordan	1	1	0	NO

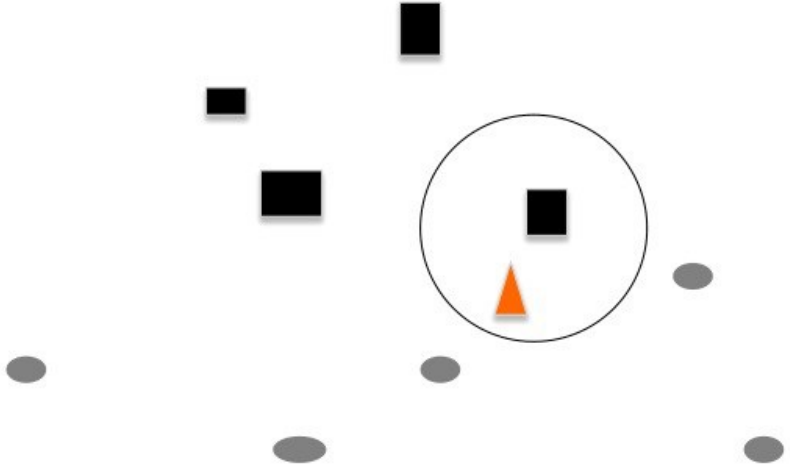
Each internal node tests an attribute

Each branch corresponds to an attribute value node

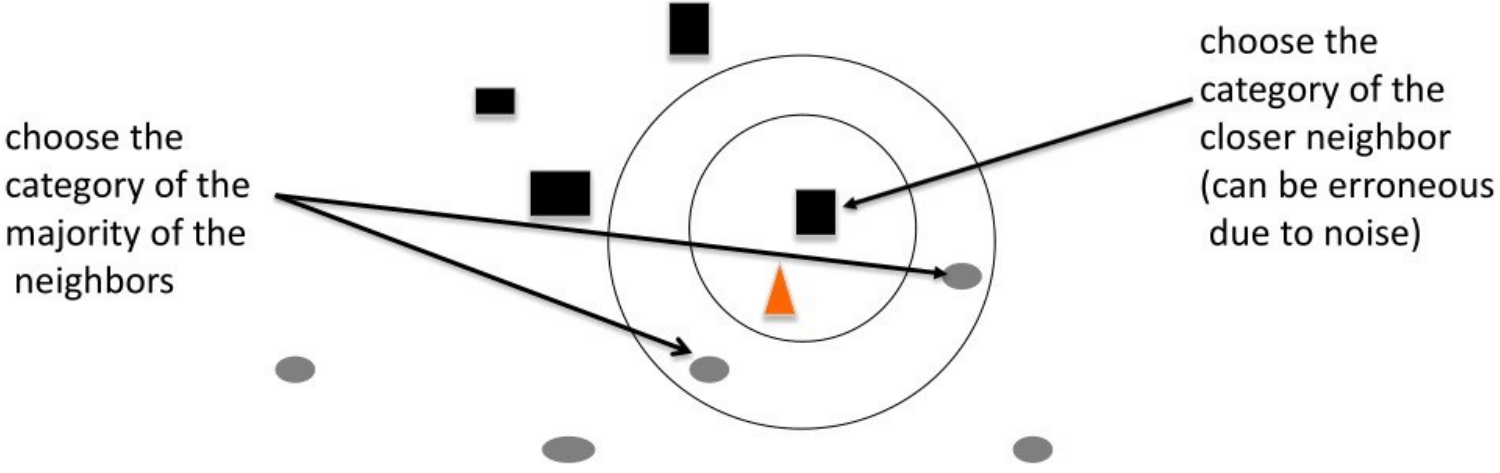
Each leaf node assigns a classification



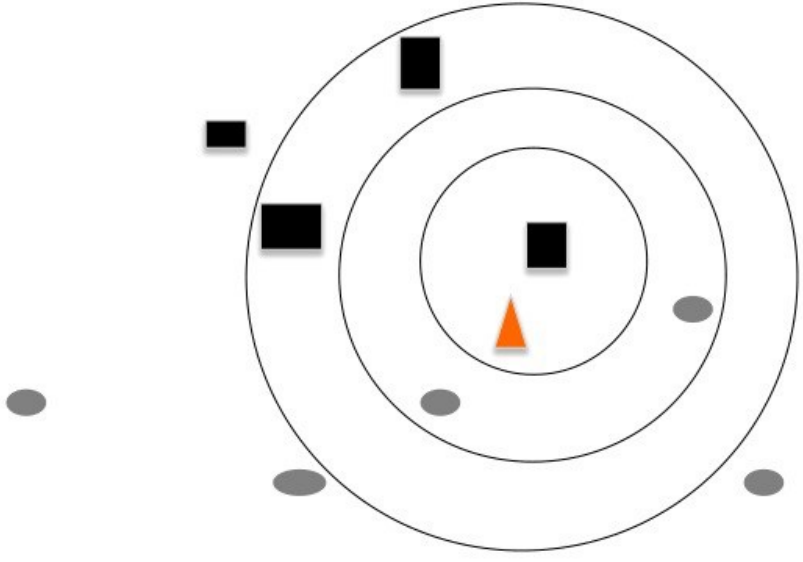
1-Nearest Neighbor



3-Nearest Neighbor



5-Nearest Neighbor



the value of *k* is typically odd to avoid ties

k Nearest Neighbours

Pros

+ robust
+ simple
+ training is very fast (storing examples)

Cons

- depends on similarity measure & k-NNs
- easily fooled by irrelevant attributes
- computationally expensive

Decision Trees

Pros

+ generate understandable rules

+ provide a clear indication of which features are most important for classification

Cons

- error prone in multi-class classification and small number of training examples

- expensive to train due to pruning

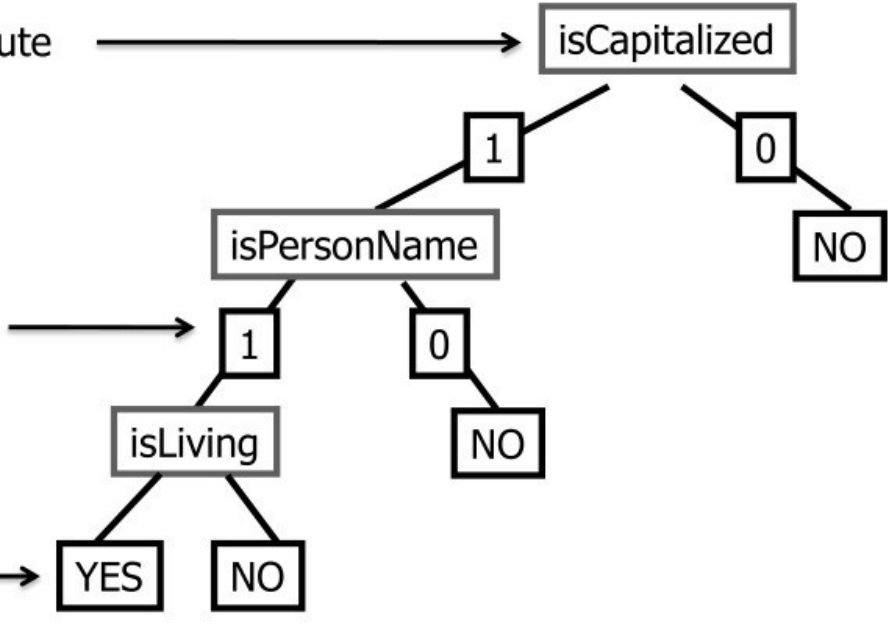
An Example

	isPersonNam e	isCapitalized	isLiving	X is PersonName?
profession	0	0	0	NO
Jerry Hobbs	1	1	1	YES
USC	0	1	0	NO
Jordan	1	1	0	NO

Each internal node tests an attribute

Each branch corresponds to an attribute value node

Each leaf node assigns a classification



Building Decision Trees

- Select which attribute to test at each node in the tree.
- The goal is to select the attribute that is most useful for classifying examples.
- Top-down, greedy search through the space of possible decision trees. It picks the best attribute and never looks back to reconsider earlier choices.

Decision Trees

Pros

+ generate understandable rules

+ provide a clear indication of which features are most important for classification

Cons

- error prone in multi-class classification and small number of training examples

- expensive to train due to pruning

Carreras et al. 2002

- Learning algorithm: AdaBoost
- Binary classification
- Binary features

- $f(x) = \sum_{t=1}^{T} \alpha_t h_t(x)$ (Schapire & Singer, 99)
- Weak rules (h_t): Decision Trees of fixed depth.

Features for NE Detection (1)

Adam	Adam, null, null, null, Smith, works, for
Smith	Smith, Adam, null, null, works, for, IBM
Works	
for	
IBM	.....
in	
London	
.	fp, London, in, IBM, null, null, null

- **Contextual**
  - current word W_0
  - words around W_0 in [-3,…,+3] window

Features for NE Detection (3)

- **Orthographic (binary and not mutually exclusive)**

initial-caps	*all-caps*	*all-digits*
roman-number	*contains-dots*	*contains-hyphen*
acronym	*lonely-initial*	*punctuation-mark*
single-char	*functional-word**	*URL*

- **Word-Type Patterns:**

functional	*lowercased*	*quote*
capitalized	*punctuation mark*	*other*

- **Left Predictions**
  - the tag predicted in the current classification for W_{-3}, W_{-2}, W_{-1}
- **Part-of-speech tag** (when available)

The more useful features you incorporate, the more powerful your learner gets

**functional-word is preposition, conjunction, article*

Results for NE Detection

Carreras et al.,2002	Precision	Recall	F-score
BIO dev.	92.45	90.88	91.66

CoNLL-2002 Spanish Evaluation Data		
Data sets	#tokens	#NEs
Train	264,715	18,794
Development	52,923	4,351
Test	51,533	3,558

Evaluation Measures