Natural Language Understanding

CSCI 4907/6515

Lecture 2: January 20, 2026

Aya Zirikly

Types and tokens

- Type = abstract descriptive concept
- Token = instantiation of a type

To be or not to be

6 tokens (to, be, or, not, to, be)
4 types (to, be, or, not)

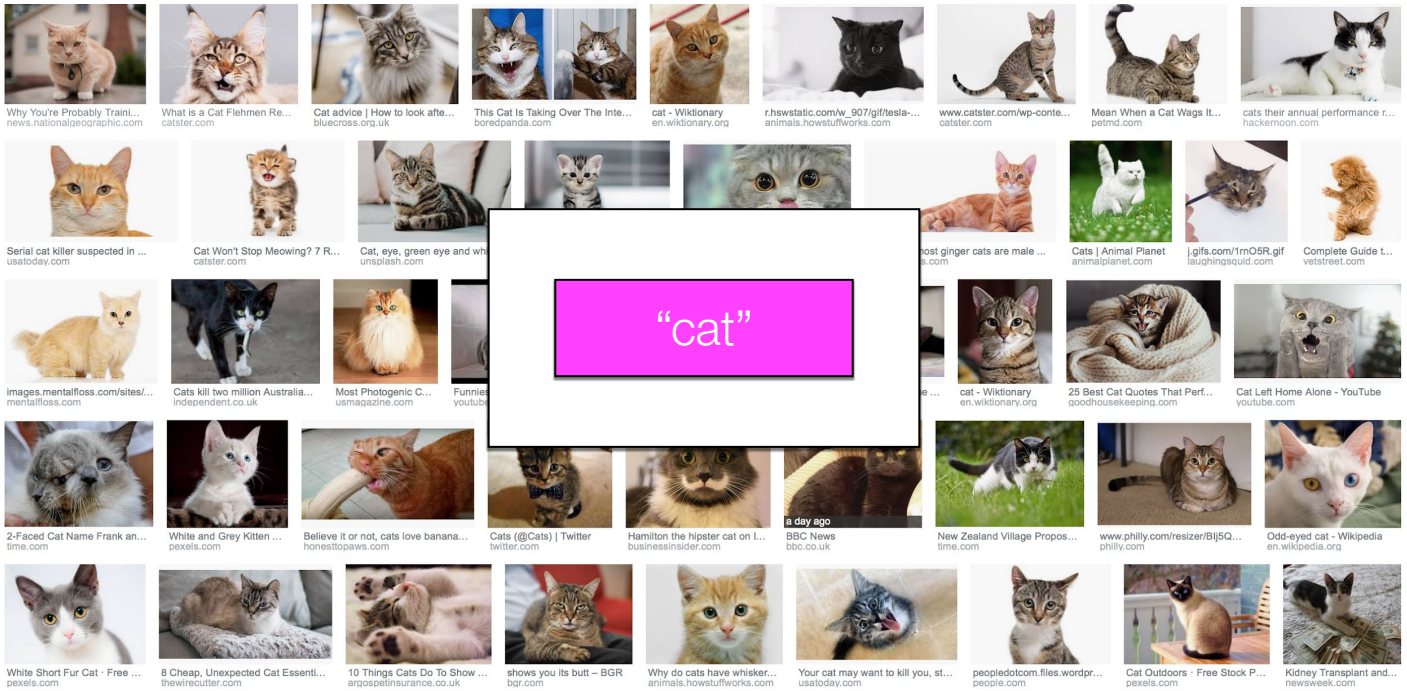
- Types = the vocabulary; the unique tokens.

Types and tokens

- Type = abstract descriptive concept
- Token = instantiation of a type

How can we use types and tokens to measure vocabulary richness (lexical diversity)?

Words as dimensionality reduction



https://brenocon.com/blog/2012/07/the-60000-cat-deep-belief-networks-make-less-sense-for-language-than-vision/

Words

- One morning I shot an elephant in my pajamas
- I didn’t shoot an elephant
- Imma let you finish but Beyonce had one of the best videos of all time
- I do uh main- mainly business data processing
- 一天早上我穿着睡衣射了一只大象

Words

A linguistic unit of meaning

"Ice cream" vs. "Sun"

- **"Sun"** is 1 concept, 1 token.
- **"Ice cream"** is 1 concept, but **2 tokens** if we just split by space.
- **"High school"** is 1 concept, **2 tokens**.

- Should "Ice cream" count as two separate words ("Ice" + "Cream") or one single vocabulary item?

"Don't"

- If we split by space, it is **1 token**: `["Don't"]`.
- But grammatically, it is **2 words**: "Do" + "Not".

NLP Question: If we keep it as one token, the computer won't know it contains the word "not" (negation). How do we split it? `["Do", "n't"]`? `["Don", "'t"]`?

Tokenization

I’m going to Washington (D.C.) → I ’m going to Washington (D.C.)

Tokenization

@Aya have you seen this :)
http://cs.gwu.edu

→

@
Aya
have
you
seen
this
:
)
http://cs.gwu.edu

Whitespace

```
text.split(“ ”)
```

- As much mud in the streets as if the waters had but newly retired from the face of the earth, and it would not be wonderful to meet a Megalosaurus, forty feet long or so, waddling like an elephantine lizard up Holborn Hill.

Whitespace

```
text.split(“ ”)
```

- As much mud in the streets as if the waters had but newly retired from the face of the earth, and it would not be wonderful to meet a Megalosaurus, forty feet long or so, waddling like an elephantine lizard up Holborn Hill.

what do we lose with whitespace tokenization?

368	earth
135	earth,
68	earth.
26	earth
24	earth.
18	earth."
16	earth;
14	earth,
9	earth's
5	earth!"
5	earth!
4	earth;
4	earth,"
3	earth.”
3	earth?
3	earth!"

2	earth--to
2	earth--if
2	earth--and
2	earth:
2	earth,'
1	earth-worms,
1	earth-worm.
1	earth--which
1	earth--when
1	earth--something
1	earth-smeared,
1	earth-scoops,
1	earth's
1	earth--oh,

“earth” in sample of 100 books (Project Gutenberg)

Punctuation

- We typically don’t want to just strip all punctuation, however.
    - Punctuation signals boundaries (sentence, clausal boundaries, parentheticals, asides)
    - Some punctuation has illocutionary force, like exclamation points (!) and question marks (?)
    - Emoticons are strong signals of e.g. sentiment

Tokenization

- Top-down tokenizers (for languages delimited by whitespace) use regular expressions to segment a text string into tokens through a sequence of regex disjunctions, each matching specific patterns that constitute a “word”

Regular expressions

A language for specifying search strings in text.

`/waters/`

> As much mud in the streets as if the waters had but newly retired from the face of the earth, and it would not be wonderful to meet a Megalosaurus, forty feet long or so, waddling like an elephantine lizard up Holborn Hill.

Charles Dickens' novel Bleak House, Kleene 1951

Kleene’s Theorem

If you can build a finite machine to recognize a pattern, you can write a regular expression for it. Conversely, if you can write a regular expression, you can build a machine to recognize it.

He introduced the three fundamental operations we still use in Python, Java, and grep today:

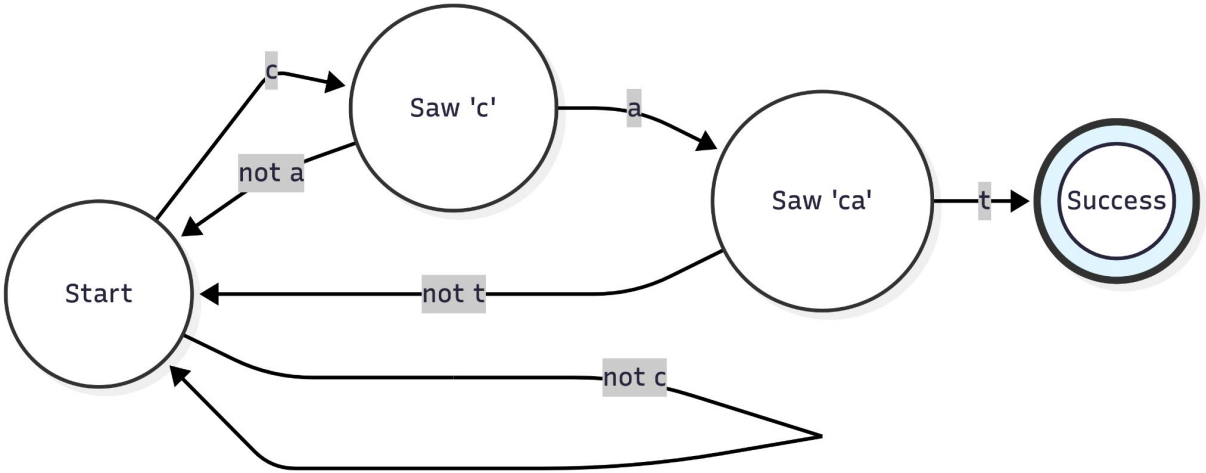
- Union (+ or |): "A or B" (e.g., cat|dog).
- Concatenation (.): "A followed by B" (e.g., cat followed by s).
- The Kleene Star (*): The most famous contribution. It represents "zero or more repetitions" of a pattern. This was the mathematical tool needed to describe infinite loops in finite machines.

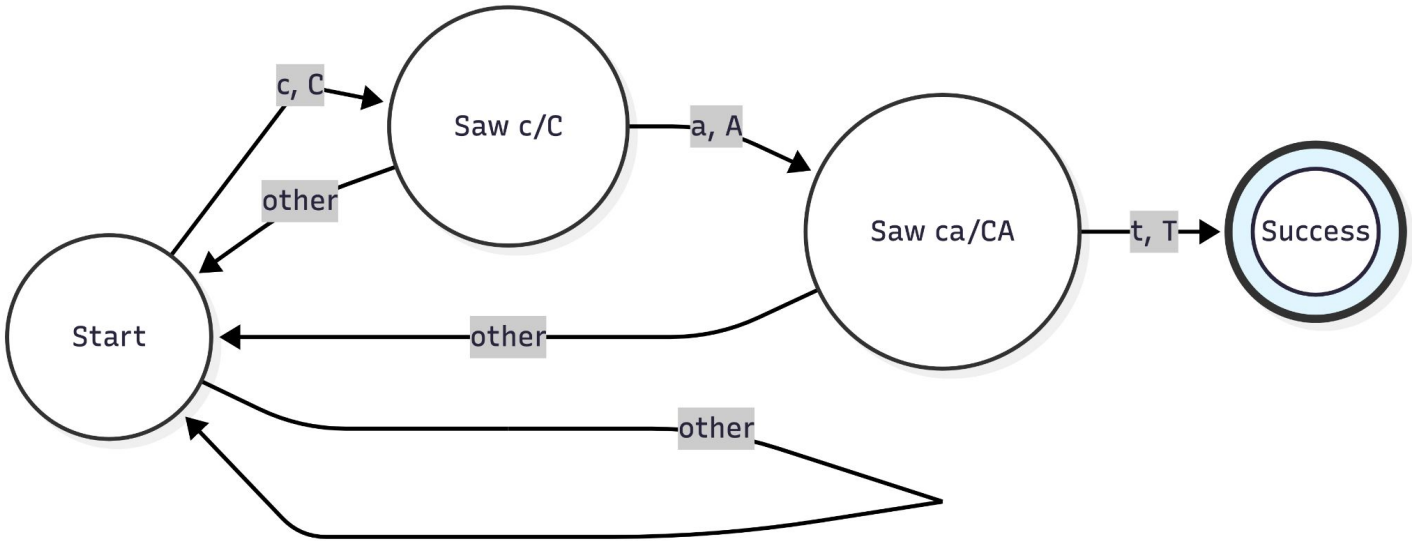
Kleene’s Theorem

Before Kleene: We knew machines could calculate, but we didn't have a formal language to describe *patterns* of symbols.

After Kleene: We knew that simple patterns (like email addresses, dates, or word suffixes) could be recognized by very efficient, simple machines (Finite State Automata) without needing the power of a full human brain or a Turing machine.

Finite State Automaton to recognize cat




c, C
Saw c/C
a, A
other
Saw ca/CA
t, T
Success
Start
other
other

Regular expressions

Term	Meaning	Sample regex	Matches
+	one or more	he+y	hey, heeeeeey
?	optional	colou?r	color, colour
*	zero or more	toys*	toy, toys, toysss

Regular expressions

A language for specifying search strings in text

`/ing?/`

As much mud in the streets as if the waters had but newly retired from the face of the earth, and it would not be wonderful to meet a Megalosaurus, forty feet long or so, waddling like an elephantine lizard up Holborn Hill.

Disjunction

- We can specify complex regular expressions by joining separate regexes with a disjunction operator |

`/(waters?)|(earth)|([Hh]ill)/`

As much mud in the streets as if the waters had but newly retired from the face of the earth, and it would not be wonderful to meet a Megalosaurus, forty feet long or so, waddling like an elephantine lizard up Holborn Hill.

Regular expressions

```
>>> text = 'That U.S.A. poster-print costs $12.40...'
>>> pattern = r'''(?x)    # set flag to allow verbose regexps
...     (?:[A-Z]\.)+        # abbreviations, e.g. U.S.A.
...   | \w+(?:-\w+)*        # words with optional internal hyphens
...   | \$?\d+(?:\.\d+)?%?  # currency, percentages, e.g. $12.40, 82%
...   | \.\.\.              # ellipsis
...   | [][.,;"'?():_`-]  # these are separate tokens; includes ], [
... '''
>>> nltk.regexp_tokenize(text, pattern)
['That', 'U.S.A.', 'poster-print', 'costs', '$12.40', '...']
```

```
import nltk
tokens=nltk.word_tokenize(text)
```

Tokenizes following the conventions of the Penn Treebank:

- punctuation split from adjoining words
- double quotes (“) changes to forward/backward quotes based on on their location in word (`the’’)
- verb contractions + ’s split into separate tokens: (`did_n’t`, `children_’s`)

ftp://ftp.cis.upenn.edu/pub/treebank/public_html/tokenization.html

```
import nltk
tokens=nltk.word_t
```

Tokenizes following the conventions of th

- punctuation split from adjoining w
- double quotes (“) changes to forw ir location in word (`the’’)
- verb contractions + ’s split into se `children_'s`)

The **Penn Treebank** is a seminal dataset of text (primarily from 1989 *Wall Street Journal* articles) in which every word is manually tagged with its part of speech and every sentence is mapped into a grammatical tree structure.

It serves as the standard "answer key" for computer scientists, used to train and test how well natural language processing (NLP) models understand English syntax.

ftp://ftp.cis.upenn.edu/pub/treebank/public_html/tokenization.html

```
import nltk
tokens=nltk.word_tokenize(text)
```

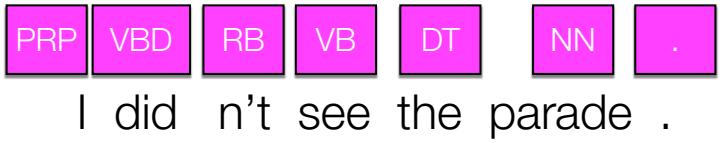
Tokenizes following the conventions of the Penn Treebank:

- punctuation split from adjoining words
- double quotes (“) changes to forward/backward quotes based on on their location in word (`the’’)
- verb contractions + ’s split into separate tokens: (`did_n’t`, `children_’s`)

ftp://ftp.cis.upenn.edu/pub/treebank/public_html/tokenization.html

```
import nltk
tokens=nltk.word_tokenize(text)
```

Penn Treebank tokenization is important because a lot of downstream NLP is trained on annotated data that uses Treebank tokenization!



Clitic split

Tokenization

- Bottom-up tokenizers learn subword tokens based on statistical regularities in the data. Because they are learned, the tokenizer is dependent on the data used to learn from.

Subtokenization

- The set of unique words (the vocabulary) in a language can number in the hundreds of thousands (e.g. unix `dict/words` has 236K words).

Agglutinative languages

Morpheme	Meaning	Grammatical Function
Muvaffak	Successful	Root (Adjective)
-iyet	-ness / State of	Turns Adjective → Noun ("Success")
-siz	-less / Without	Turns Noun → Adjective ("Unsuccessful")
-leş	To become	Turns Adjective → Verb ("To become unsuccessful")
-mek	To	Infinitive Marker (like "to walk")

Agglutinative languages

Muvaffakiyetsizleş(-mek)	(To) become unsuccessful
Muvaffakiyetsizleştir(-mek)	(To) make one unsuccessful
Muvaffakiyetsizleştirici	Maker of unsuccessful ones
Muvaffakiyetsizleştiricileş(-mek)	(To) become a maker of unsuccessful ones
Muvaffakiyetsizleştiricileştir(-mek)	(To) make one a maker of unsuccessful ones
Muvaffakiyetsizleştiricileştiriver(-mek)	(To) easily/quickly make one a maker of unsuccessful ones
Muvaffakiyetsizleştiricileştiriverebil(-mek)	(To) be able to make one easily/quickly a maker of unsuccessful ones
Muvaffakiyetsizleştiricileştiriveremeyebileceklerimizdenmişsinizcesine	As though you happen to have been from among those whom we will not be able to easily/quickly make a maker of unsuccessful ones

Agglutinative langua

Challenge: You cannot train a model on "vocabulary." You must train it on "morphemes" (the sub-parts).

Muvaffakiyetsizleş(-mek)	(To) become unsuccess
Muvaffakiyetsizleştir(-mek)	(To) make one unsuccessful
Muvaffakiyetsizleştirici	Maker of unsuccessful ones
Muvaffakiyetsizleştiricileş(-mek)	(To) become a maker of unsuccessful ones
Muvaffakiyetsizleştiricileştir(-mek)	(To) make one a maker of unsuccessful ones
Muvaffakiyetsizleştiricileştiriver(-mek)	(To) easily/quickly make one a maker of unsuccessful ones
Muvaffakiyetsizleştiricileştiriverebil(-mek)	(To) be able to make one easily/quickly a maker of unsuccessful ones
Muvaffakiyetsizleştiricileştiriveremeyebileceklerimizdenmişsinizcesine	As though you happen to have been from among those whom we will not be able to easily/quickly make a maker of unsuccessful ones

Infinitive
danser

Past Participle
dansé

Gerund
dansant

Imperative
danse (tu)
dansons (nous)
dansez (vous)

Present
je danse
tu danses
il/elle danse
nous dansons
vous dansez
ils/elles dansent

Present Perfect
j'ai dansé
tu as dansé
il/elle a dansé
nous avons dansé
vous avez dansé
ils/elles ont dansé

Imperfect
je dansais
tu dansais
il/elle dansait
nous dansions
vous dansiez
ils/elles dansaient

Future
je danserai
tu danseras
il/elle dansera
nous danserons
vous danserez
ils/elles danseront

Conditional
je danserais
tu danserais
il/elle danserait
nous danserions
vous danseriez
ils/elles danseraient

Past Historic
je dansai
tu dansas
il/elle dansa
nous dansâmes
vous dansâtes
ils/elles dansèrent

Pluperfect
j'avais dansé
tu avais dansé
il/elle avait dansé
nous avions dansé
vous aviez dansé
ils/elles avaient dansé

Future Perfect
j'aurai dansé
tu auras dansé
il/elle aura dansé
nous aurons dansé
vous aurez dansé
ils/elles auront dansé

Past Anterior
j'eus dansé
tu eus dansé
il/elle eut dansé
nous eûmes dansé
vous eûtes dansé
ils/elles eurent dansé

Conditional Perfect
j'aurais dansé
tu aurais dansé
il/elle aurait dansé
nous aurions dansé
vous auriez dansé
ils/elles auraient dansé

Present Subjunctive
je danse
tu danses
il/elle danse
nous dansions
vous dansiez
ils/elles dansent

Imperfect Subjunctive
je dansasse
tu dansasses
il/elle dansât
nous dansassions
vous dansassiez
ils/elles dansassent

Present Perfect Subjunctive
j'aie dansé
tu aies dansé
il/elle ait dansé
nous ayons dansé
vous ayez dansé
ils/elles aient dansé

Pluperfect Subjunctive
j'eusse dansé
tu eusses dansé
il/elle eût dansé
nous eussions dansé
vous eussiez dansé
ils/elles eussent dansé

https://www.collinsdictionary.com/dictionary/french-english/conjugation/danser

Subtokenization

- Sometimes it's useful to decompose words into subunits ("subwords") in order to expose structure ***within*** a token.

Subtokenization

- supercalifragilisticexpialidocious
    - super, superior, supernatural
    - adventurous, fabulous, infamous
- supercalifragilisticexpialidociously
    - quickly, sadly, perfectly

Byte-Pair Encoding

- Learn a compressed tokenization to encode frequently-occurring sequences of characters as their own tokens.
- Sennrich et al. (2016), “Neural Machine Translation of Rare Words with Subword Units”

Byte-Pair Encoding

corpus

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

vocabulary

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

Byte-Pair Encoding

corpus

5	l o w _
2	l o w e s t _
6	n e w er _
3	w i d er _
2	n e w _

vocabulary

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

Byte-Pair Encoding

corpus

5	l o w _
2	l o w e s t _
6	n e w er_
3	w i d er_
2	n e w _

vocabulary

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

Byte-Pair Encoding

corpus

5 l o w _
2 l o w e s t _
6 ne w er_
3 w i d er_
2 ne w _

vocabulary

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

Byte-Pair Encoding

merge	current vocabulary
(ne, w)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new
(l, o)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo
(lo, w)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo, low
(new, er_)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo, low, newer_
(low, _)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo, low, newer_, low_

Byte-Pair Encoding

- To tokenize a new string used BPE, we apply the merge rules greedily in the same order in which they were originally learned.

"My name is Inigo Montoya"

Keep common words whole, but break rare words into chunks.

Byte-Pair Encoding

"My name is Inigo Montoya"

Word	Status	BPE Tokens	Why?
My	Common	[My]	Frequent word. Kept whole.
name	Common	[name]	Frequent word. (Note: The leading space is usually included).
is	Common	[is]	Frequent word.
Inigo	Rare	[In], [igo]	"In" is a very common prefix. "igo" is a common suffix.
Montoya	Rare	[Mont], [oya]	"Mont" (like Vermont/Montana) is common. "oya" (like Toyota) is common.

Subtokenization

- The set of unique words (the vocabulary) in a language can number in the hundreds of thousands (e.g. unix `dict/words` has 236K words).
- BPE encoding (and other forms of vocabulary compression) allow us to bound the size of a vocabulary (e.g., 30K words) while also surfacing subword structure that can be useful for downstream models.

BPE

Model	Vocabulary Size	Tokenizer Name	Efficiency
GPT-2 / GPT-3	50,257 tokens	r50k_base	Good, but verbose with code/numbers.
GPT-4	100,277 tokens	cl100k_base	**Much better.** It is better at merging numbers (e.g., "123") and coding brackets into single tokens.

GPT-5.x & O1/3 | **GPT-4 & GPT-3.5 (legacy)** | GPT-3 (legacy)

Many words map to one token, but some don't: indivisible.

Unicode characters like emojis may be split into many tokens containing the underlying bytes: 🤚

Sequences of characters commonly found next to each other may be grouped together: 1234567890

Clear | Show example

Tokens 57

Characters 252

Many words map to one token, but some don't: indivisible.

Unicode characters like emojis may be split into many tokens containing the underlying bytes: ??????

Sequences of characters commonly found next to each other may be grouped together: 1234567890

Text | Token IDs

Tiktoken

TokenIDs

```
[8607, 4339, 2472, 311,
832, 4037, 11, 719, 1063,
1541, 956, 25, 3687, 23936,
382, 35020, 5885, 1093,
100166, 1253, 387, 6859,
1139, 1690, 11460, 8649,
279, 16940, 5943, 25,
11410, 97, 248, 9468, 237,
122, 271, 1542, 45045, 315,
5885, 17037, 1766, 1828,
311, 1855, 1023, 1253, 387,
41141, 3871, 25, 220, 4513,
10961, 16474, 15]
```

https://platform.openai.com/tokenizer

Sentence segmentation

- Word tokenization presumes a preprocessing step of sentence segmentation — identifying the boundaries between sentences.
- Lots of NLP operates at the level of the sentence (POS tagging, parsing), so really important to get it right.
- Harder to write regexes to delimit these, since there are many cases where the usual delimiters (periods, question marks) serve double duty.

Sentence segmentation

- “Do you want to go?” said Jane.
- Mr. Collins said he was going.
- He lives in the U.S. John, however, lives in Canada.

Sentence segmentation

- NLTK: Punkt sentence tokenizer — unsupervised method to learn common abbreviations, collocations, sentence-initial words. Can be trained on data from new domain.
  [Kiss, Tibor and Strunk, Jan (2006): Unsupervised Multilingual Sentence Boundary Detection (*Computational Linguistics*)]

- spaCy: Relies on dependency parsing to find sentence boundaries.

```
import spacy
nlp = spacy.load('en_core_web_sm')
doc=nlp(text)
for sent in doc.sents:
    for token in sent:
        print(token.text)
```

Sentence segmentation -before and now

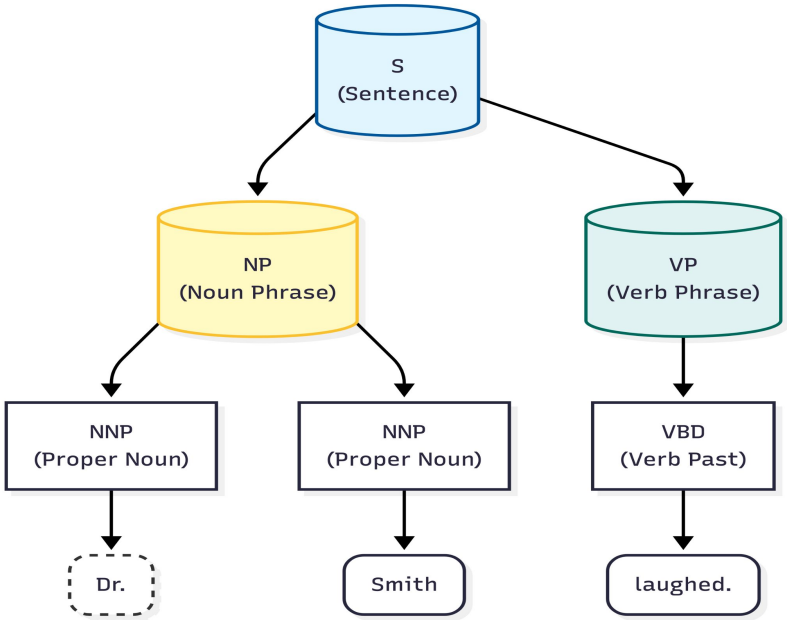
Feature	Traditional NLP (e.g., spaCy/NLTK)	ChatGPT (LLM)
Unit of Analysis	The Sentence	The Token (Subword)
Method	Explicit Rule/Model (Sentence Segmenter)	Implicit Probability (Next-Token Prediction)
Handling "Dr."	Often breaks (requires exception lists)	Handles naturally via context
Structure	Rigid (Sentence 1 → Sentence 2)	Fluid (Continuous stream)

Sentence segmentation -before and now

Feature	ChatGPT (Decoder)	BERT (Encoder)
View of Text	Continuous River	Discrete Pairs (A and B)
Sentence Boundary	Implicit (Predicts .)	**Explicit** (Uses [SEP] token)
Preprocessing	Minimal (Raw text in)	**High** (Must split sentences & add special tokens)
Structural Knowledge	Learned from context	Hard-coded via "Segment Embeddings"

Sentence segmentation -using parsing

Dr. Smith laughed.



Stemming and lemmatization

- Many languages have some inflectional and derivational morphology, where similar words have similar forms:

  organizes, organized, organizing

- Stemming and lemmatization reduce this variety to a single common base form.

Stemming

- Heuristic process for chopping off the inflected suffixes of a word

organizes, organized, organizing → organ

Porter stemmer

- Sequence of rules for removing suffixes from words

- EMENT → ∅ replacement → replac
- SSES → SS dresses → dress
- IES → I flies → fli
- SS → SS pass → pass
- S → ∅ cats → cat

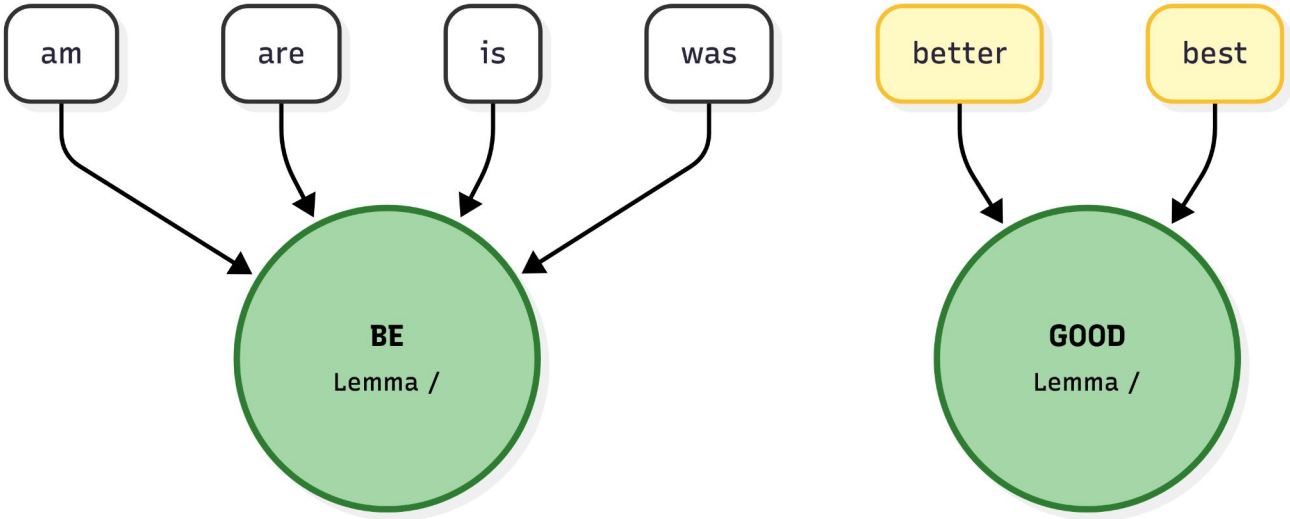
Lemmatization

- Using morphological analysis to return the dictionary form of a word (the entry in a dictionary you'd find all forms under)

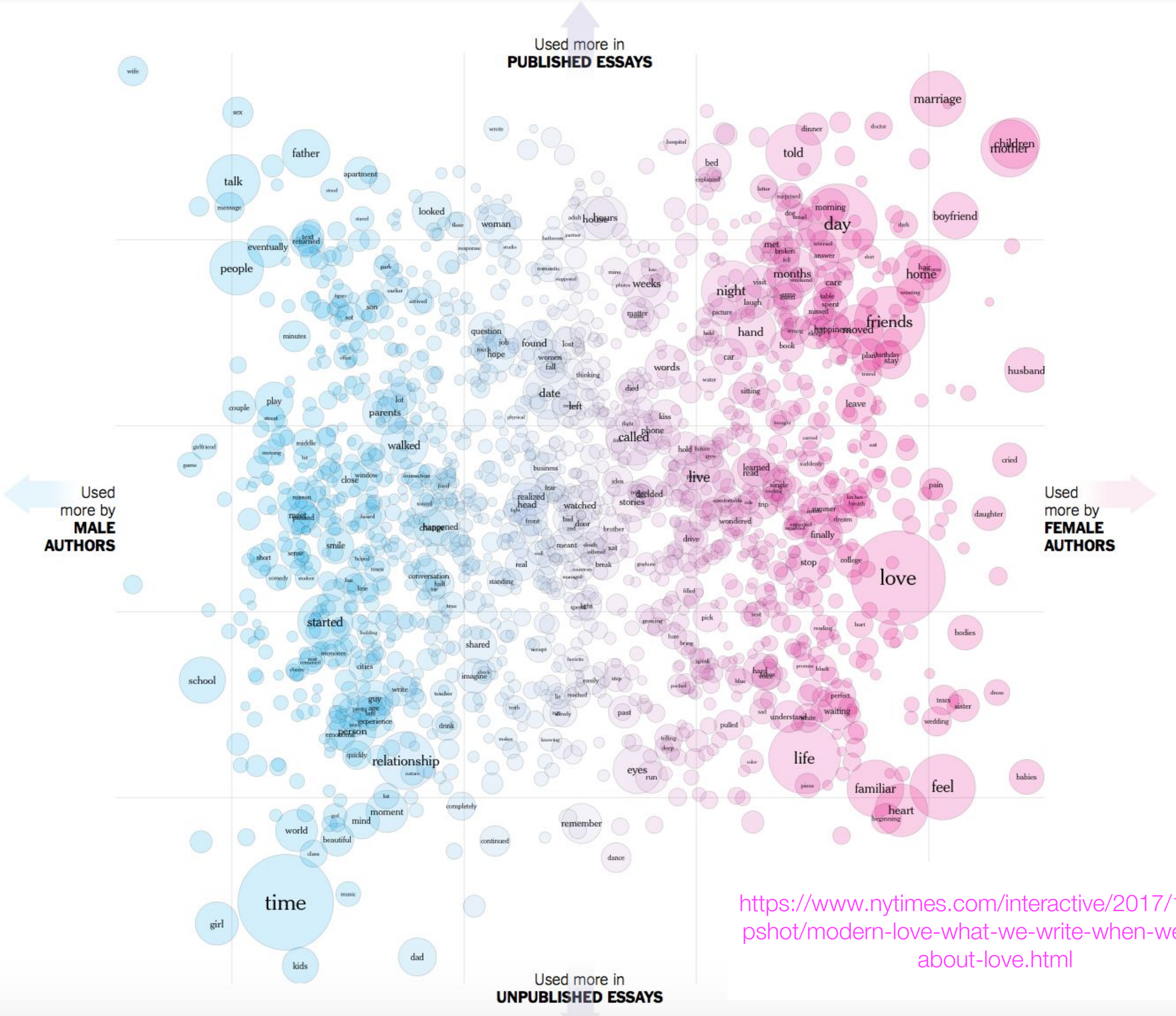
organizes, organized, organizing → organize

```
import spacy
nlp = spacy.load('en_core_web_sm')
lemmas=[token.lemma_ for token in nlp(text)]
```

Lemmatization



What can we do with just words?



https://www.nytimes.com/interactive/2017/11/07/upshot/modern-love-what-we-write-when-we-write-about-love.html

Dataset	Top verbs for $S_{\uparrow}$ ($A > 1$)	Top verbs for $S_{\downarrow}$ ($A < -1$)
arXiv	achieve, **learn**, **guide**, show, embed, **fool**, find, **need**, **assist**, follow, **search**, **mislead**, inspire, win, demonstrate, **benefit**, try, **face**, deceive, plan, **make**, **steer**, generative, attempt, **retrain**, **train**, flow, weight, **require**, alternate, focus, **motivate**, experiment, tackle, **see**, hide, spiking, recommend, **discover**, participate, spike, **pass**, code, check, suggest, **decide**, interference, aim, move	**propose**, **present**, **outperform**, **develop**, **be**, **evaluate**, **improve**, **introduce**, **allow**, **use**, **compare**, **extend**, **implement**, give, **apply**, **consist**, **validate**, design, **yield**, analyze, **combine**, test, **leverage**, **deploy**, adapt, **build**, generalize, **enhance**, **devise**, **become**, optimize, reduce, derive, **utilize**, scale, study, **run**, modify, converge, illustrate, assess, **increase**, provide, contain, surpass, maximize, perform, complement, depend, simplify
News	say, hire, beat, encounter, fool	develop, use, build, be, create, introduce, help
ACL (unique)	provide, have, generate, create, parse, enable, suffer, construct, capture, obtain, fail, encourage, struggle, understand, help, do, select, extract, tend, predict, training, handle, lack, encode, deal, identify, ask, prevent, distinguish, model, establish, respond, ignore, report, inform, choose, interpret, recurrent, detect, seem	achieve, rely, explore, employ, show, adopt, investigate, include, demonstrate, submit, integrate, prove, augment, involve, participate, aim, tune, conduct

Cheng et al. 2024, "ANTHROSCORE: A Computational Linguistic Measure of Anthropomorphism" (EACL)

Distinctive terms

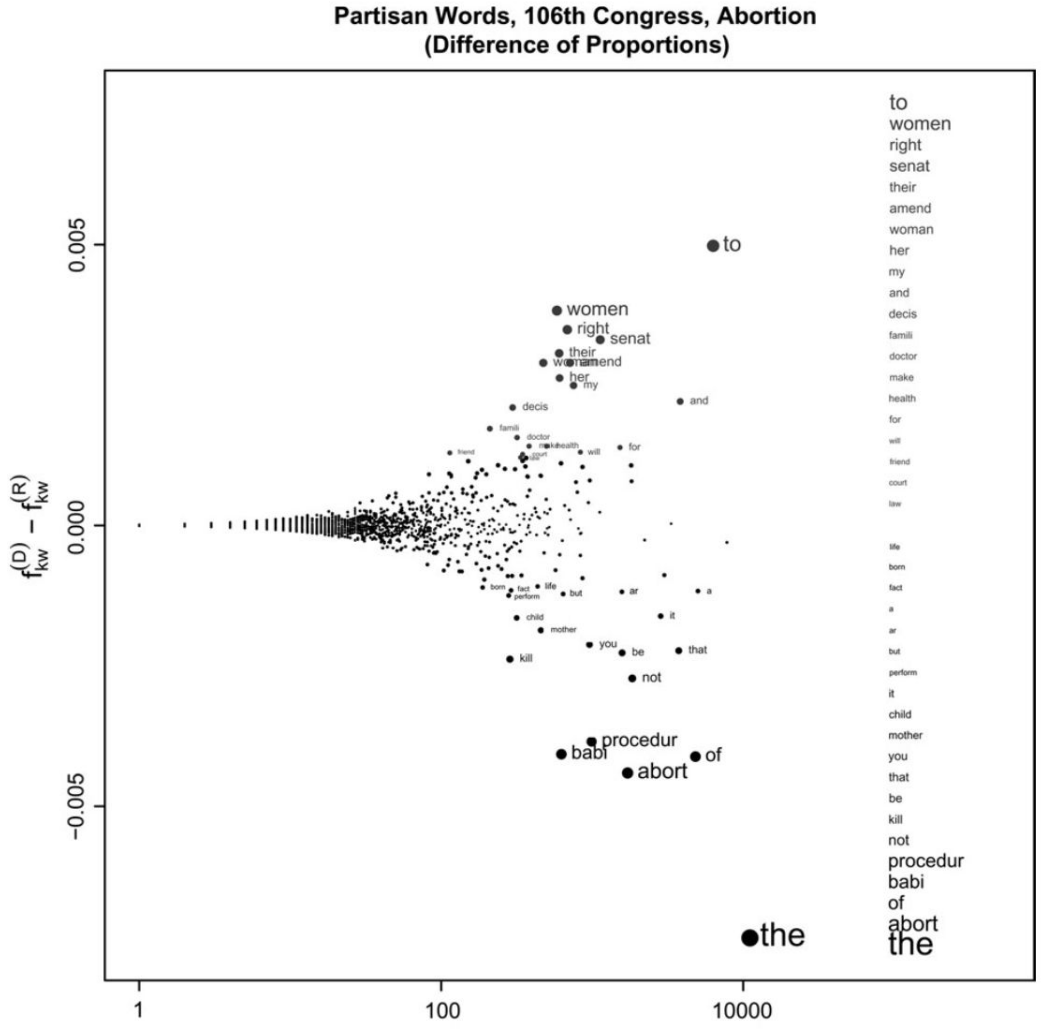
- Finding distinctive terms is useful:
    - As a data exploration exercise to understand larger trends in individual word differences).
    - As a pre-processing step of feature selection.
- When the two datasets are A and ¬A, these terms also provide insight into what A is about.
- Many methods for finding these terms! (Developed in NLP, corpus linguistics, political science, etc.)

Difference in proportions

For word w written by author with label k (e.g., {democrat, republican}), define the frequency to be the normalized count of that word

count of word w in group k

count of all words in group k



Monroe et al. (2009), “Fightin’ Words”

Difference in proportions

- The difference in proportions is a conceptually simple measure and easily interpretable.
- Drawback: tends to emphasize words with high frequency (where even comparatively small differences in word usage between groups is amplified).
- Also, no measure whether a difference is statistically meaningful. We have uncertainty about the what the true proportion is for any group.

Which words actually matter?

The Challenge:

If we want to build a system to detect "Spam" vs. "Ham" (Normal) emails, how do we choose which words to look at?

- Count the most frequent words.
  - *Result:* "The", "Is", "A", "And". (These appear in both Spam and Ham → useless
- Find words that are **dependent** on the category.

The Solution: Chi-Square (χ2)

- Statistical test of **independence**: between the two variables of word identity and corpus identity.
- It asks a simple question: **"Does the word 'Winner' appear more often in Spam than we would expect by random chance?"**
  - If **Yes**: It is a good feature. Keep it.
  - If **No**: It is noise. Throw it away.

χ^2
observed vs. expected

We compare two values:

1. **Observed Count (O):** How many times did we *actually* see the word "Prize" in Spam emails?
2. **Expected Count (E):** If words were scattered randomly (like sprinkling salt), how many times *should* "Prize" appear in Spam?

The Rule:

- If **Observed ~ Expected**: The word is boring. It has no predictive power. (Low χ2)
- If **Observed >> Expected**: The word is "sticky" to that category. It is a strong keyword. (High χ2)

$$\chi^2 = \sum \frac{(Observed - Expected)^2}{Expected}$$

X^2 Example

- Let's test the word **"Goal"** for a classifier distinguishing **Sports News** vs. **Politics News**.
- **The Contingency Table** Total documents: 100 (50 Sports, 50 Politics).

The Expected Count (Null Hypothesis)

If "Goal" had nothing to do with Sports, it should be evenly split.

- **Expected** for Sports = 25
- **Expected** for Politics = 25

	Sports	Politics	Total
Contains "Goal"	**40** (Observed)	**10** (Observed)	50
No "Goal"	10	40	50
Total	50	50	100

Sports: $(40 - 25)^2 / 25 = 9$
Politics: $(10 - 25)^2 / 25 = 9$
Total X^2 Score: $9 + 9 = 18$
Result: 18 is a very high score (statistically significant). The word "Goal" is NOT independent; it strongly signals "Sports."

X^2
Why do we use it in NLP?

Feature Selection

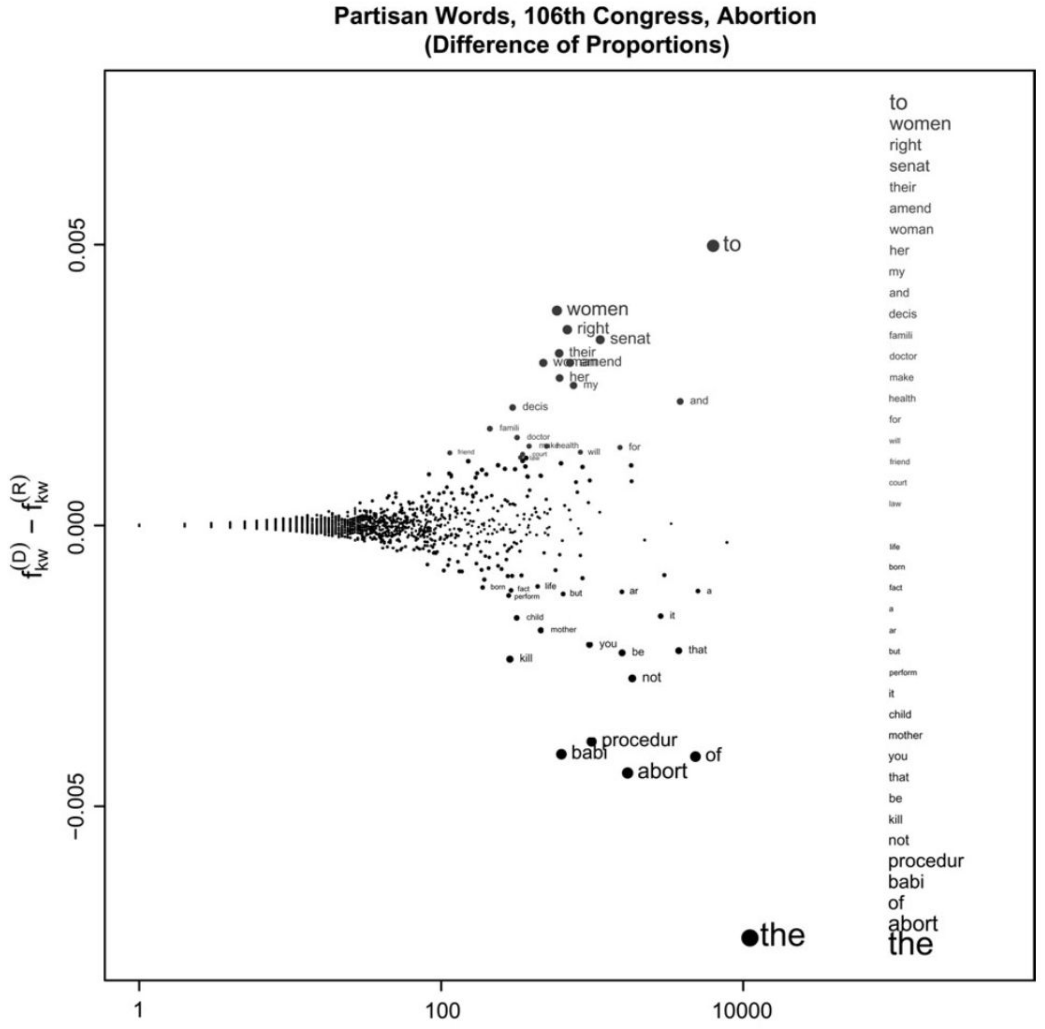
When you have a vocabulary of 50,000 words, your computer gets slow. You can calculate X^2 score for every word and keep only the top 1,000. This makes your AI faster and often more accurate because you removed the noise.

Identifying "Fightin' Words"

As seen in the Monroe et al. paper, this helps us find words that distinctively belong to one group.

- *Democrats* →"Estate Tax" (High X^2 association)
- *Republicans* →"Death Tax" (High X^2 association)

➢ Chi-Square is bad at very rare words. If a word appears only once, the math gets unstable. We usually filter out rare words before running this test.



Monroe et al. (2009), “Fightin’ Words”

Log-odds ratio

Standard tool to compare two groups because it handles comparisons much better than simple percentages.

1. Probability The probability of a Republican saying "Freedom" is 0.8 (80%).

P(Freedom) = 0.8

2. Odds The ratio of Success to Failure P / (1-P)

0.8/0.2 = 4 **Meaning:** For every 1 time they *don't* say it, they say it 4 times. The odds are "4 to 1".

3. Log-Odds

- **log(P / 1-P)**

4. Log-Odds Ratio compare two groups

Log-Odds Ratio = log(Odds for Democrats) - log(Odds for Republicans)

- **Result > 0:** The word belongs to Democrats
- **Result < 0:** The word belongs to Republicans.
- **Result ≈ 0:** The word is used equally by both.

Question

to be or not to be that is the question

Q1:
Assuming whitespace tokenization, how many word *types* are in the sentence above?

A. 4
B. 6
C. 8
D. 10

Q2:
Assuming whitespace tokenization, how many word *tokens* (or *instances*) are in the sentence above?

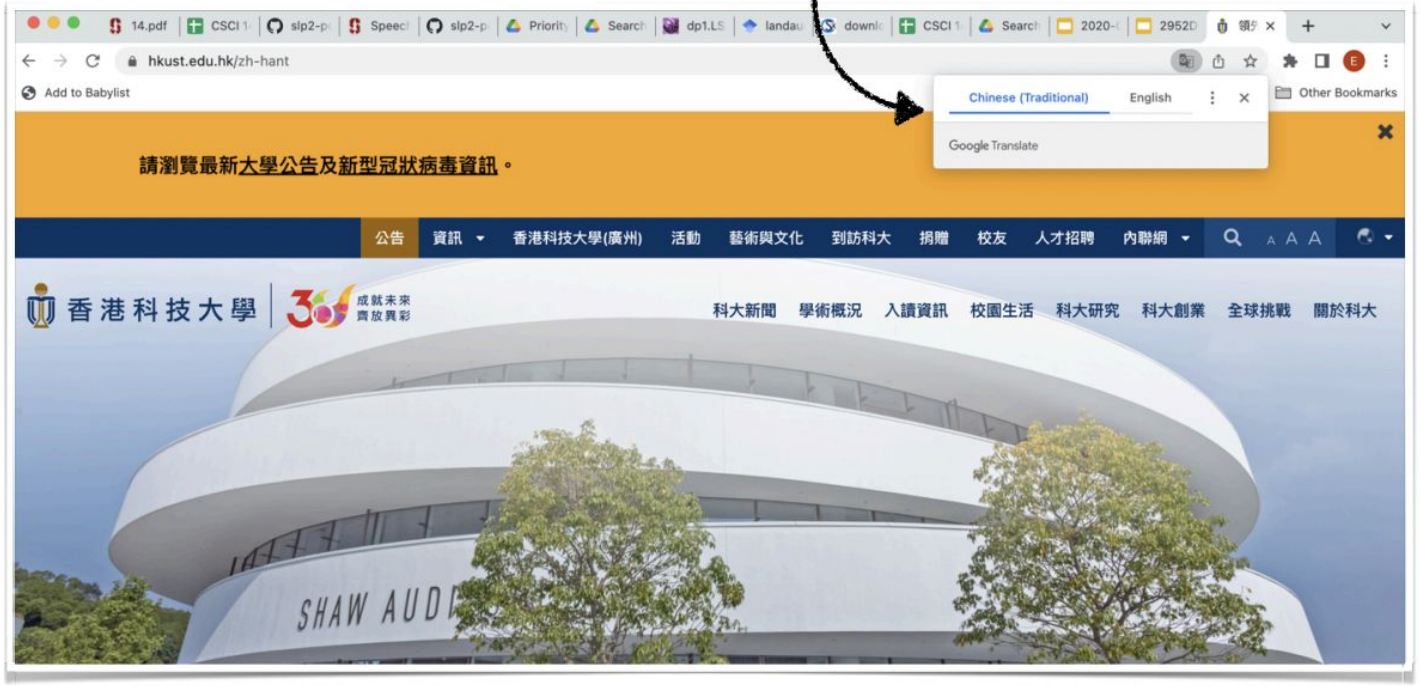
A. 4
B. 6
C. 8
D. 10

Text Classification

Naive Bayes

Text categorization/classification = The task of assigning a label or category to a text document

Language identification



Is this spam?

Spam detection

Subject: **Important notice!**
From: Stanford University <newsforum@stanford.edu>
Date: October 28, 2011 12:34:16 PM PDT
To: undisclosed-recipients:;

Greats News!

You can now access the latest news by using the link below to login to Stanford University News Forum.

http://www.123contactform.com/contact-form-StanfordNew1-236335.html

Click on the above link to login for more information about this new exciting forum. You can also copy the above link to your browser bar and login for more information
about the new services.

What is the subject of this medical article?

MEDLINE Article

Syntactic frame and verb bias in aphasia: Plausibility judgments of undergoer-subject sentences

?

MeSH Subject Category Hierarchy

Antogonists and Inhibitors

Blood Supply

Chemistry

Drug Therapy

Embryology

Epidemiology

…

Sentiment Analysis

\+ *...zany characters and richly applied satire, and some great plot twists*

\– *It was pathetic. The worst part about it was the boxing scenes...*

\+ *...awesome caramel sauce and sweet toasty almonds. I love this place!*

\– *...awful pizza and ridiculously overpriced...*

- **Text classification is ubiquitous**. Anytime you have a piece of text and want to assign it a label, you need a text classifier
  - Piece of text might be anything: article, tweet, forum post, email, doctor's note, movie script, resume, product description, call log, etc
  - Label could be anything: topic of the text, sentiment of the text, whether its spam, a medical diagnosis, a predicted cost, a hiring decision, routing of claims within an organization, etc

Text Classification: definition

Input:

- a document d
- a fixed set of classes $C = \{c_1, c_2, \ldots, c_J\}$

Output: a predicted class $c \in C$

Text Classification: definition

Input:

- a document *d*
- a fixed set of classes $C = \{c_1, c_2, \ldots, c_J\}$

Binary classification: C = {c1, c2}
vs. Multiclass classification

Output: a predicted class $c \in C$

Sentiment Analysis

\+ *...zany characters and richly applied satire, and some great plot twists*

\– *It was pathetic. The worst part about it was the boxing scenes...*

\+ *...awesome caramel sauce and sweet toasty almonds. I love this place!*

\– *...awful pizza and ridiculously overpriced...*

What are the documents?

Sentiment Analysis

\+ *...zany characters and richly applied satire, and some great plot twists*

\– *It was pathetic. The worst part about it was the boxing scenes...*

\+ *...awesome caramel sauce and sweet toasty almonds. I love this place!*

\– *...awful pizza and ridiculously overpriced...*

Sentiment Analysis

What are the documents?

And the classes?

\+ *...zany characters and richly applied satire, and some great plot twists*

\– *It was pathetic. The worst part about it was the boxing scenes...*

\+ *...awesome caramel sauce and sweet toasty almonds. I love this place!*

\– *...awful pizza and ridiculously overpriced...*

Classification Methods:
Hand-coded Rules

```
if any of {‘bad’, ‘awful’, ‘terrible’, ‘boring’} in x:
    return NEGATIVE
elif any of {‘great’, ‘fun’, ‘enjoyed’} in x:
    return POSITIVE
elif len(x) > 20 and ‘but’ in x and ‘liked’ in x[0:12]:
    return NEGATIVE
```

Classification Methods:
Hand-coded Rules

- Rules based on combinations of words or other features
- Accuracy can be high
  - If rules carefully refined
- But building and maintaining these rules is expensive and these systems are rigid

Classification Methods: Supervised Machine Learning

- Machine Learning:
  - Predicting the future based on the past
  - *Generalizing* to make predictions on new data
- Supervised machine learning:
  - Use a set of documents that are hand-labeled with their class to learn a function to map input documents to their labels

Classification Methods: Supervised Machine Learning

Input:

- a document d
- a fixed set of classes $C = \{c_1, c_2, \ldots, c_J\}$
- A training set of m hand-labeled documents $(d_1, c_1), \ldots, (d_m, c_m)$

Output:

- a learned classifier $\gamma: d \rightarrow c$

Documents
just plain boring
entirely predictable and lacks energy
no surprises and very few laughs
very powerful
the most fun film of the summer

How to get labels?

	Cat	Documents
Training	-	just plain boring
	-	entirely predictable and lacks energy
	-	no surprises and very few laughs
	+	very powerful
	+	the most fun film of the summer

	Cat	Documents
Training	-	just plain boring
	-	entirely predictable and lacks energy
	-	no surprises and very few laughs
	+	very powerful
	+	the most fun film of the summer
Test	?	predictable with no fun

How to get labels?

- Ask people to read reviews and label
- Find reviews paired with star rating

Classification Methods: Supervised Machine Learning

Many different kind of classifiers:

- Naive Bayes
- Logistic Regression
- K-Nearest Neighbors
- Decision Trees
- Neural Networks

Classification Methods: Supervised Machine Learning

- **Naive Bayes**
- Logistic Regression
- K-Nearest Neighbors
- Decision Trees
- Neural Networks

Features

- What words are in the movie review?
  - Amazing, fantastic, hilarious
  - Disappointing, long
- ...

Naive Bayes

Using Bayes' rule for classification

- Bayes' rule allows us to get posterior probability for each of a set of hypotheses given evidence

$$P(x|y) = \frac{P(y|x)P(x)}{P(y)}$$

Using Bayes' rule for classification

- Bayes' rule allows us to get posterior probability for each of a set of hypotheses given evidence

likelihood prior

posterior

$$P(x|y) = \frac{P(y|x)P(x)}{P(y)}$$

probability of evidence

Naive Bayes

	Cat	Documents
Training	-	just plain boring
	-	entirely predictable and lacks energy
	-	no surprises and very few laughs
	+	very powerful
	+	the most fun film of the summer
Test	?	predictable with no fun

Naive Bayes

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}} P(c|d)$$

Naive Bayes

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}} P(c|d)$$

$$= \underset{c \in C}{\operatorname{argmax}} \frac{P(d|c)P(c)}{P(d)}$$

Apply Bayes rule

Naive Bayes

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}} P(c|d)$$

$$= \underset{c \in C}{\operatorname{argmax}} \frac{P(d|c)P(c)}{P(d)}$$

Apply Bayes rule

$$= \underset{c \in C}{\operatorname{argmax}} P(d|c)P(c)$$

Drop denominator

Naive Bayes

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}} P(c|d)$$

$$= \underset{c \in C}{\operatorname{argmax}} \frac{P(d|c)P(c)}{P(d)}$$

Apply Bayes rule

$$= \underset{c \in C}{\operatorname{argmax}} P(d|c)P(c)$$

Drop denominator

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}} P(f_1, f_2, \ldots, f_n|c) \;\; P(c)$$

Document = set of features

Naive Bayes

$$\hat{c} = \operatorname*{argmax}_{c \in C} P(c|d)$$

$$= \operatorname*{argmax}_{c \in C} \frac{P(d|c)P(c)}{P(d)}$$

Apply Bayes rule

$$= \operatorname*{argmax}_{c \in C} P(d|c)P(c)$$

Drop denominator

$$\hat{c} = \operatorname*{argmax}_{c \in C} \overbrace{P(f_1, f_2, \ldots, f_n|c)}^{\text{likelihood}} \overbrace{P(c)}^{\text{prior}}$$

Document = set of features

Naive Bayes
Example

1. **The Bayes Part**
   - Probability for Class A (e.g., **Spam**): 0.92
   - Probability for Class B (e.g., **Ham**): 0.08
2. **Argmax Part (The Judge):**
   - Which argument (input) produced the maximum value? "Spam"

Why do we drop the denominator?

You will often see the equation written without the division by P(x) (the evidence).

$$\text{Naive Bayes} = \underset{c}{\text{argmax}} \frac{P(x|c)P(c)}{P(x)} \approx \underset{c}{\text{argmax}} \; P(x|c)P(c)$$

Why dropping denominator: Since **argmax** only cares about **ranking** (who is higher?), dividing both sides by the same number P(x) doesn't change the winner.

Naïve Bayes

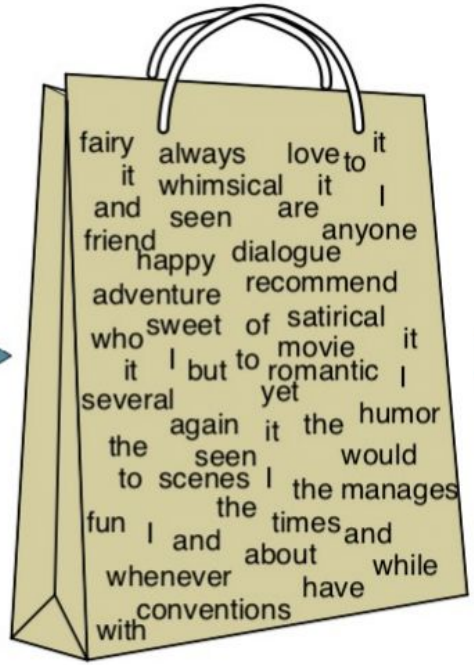
- Naïve Bayes assumption: features $f_1 \ldots f_n$ are conditionally independent given class c
- Conditional independence: $\mathbf{P}(X, Y \mid Z) = \mathbf{P}(X \mid Z)\ \mathbf{P}(Y \mid Z)$

$$P(f_1, f_2, \ldots., f_n|c) = P(f_1|c) \cdot P(f_2|c) \cdot \ldots \cdot P(f_n|c)$$

$$c_{NB} = \underset{c \in C}{\operatorname{argmax}} P(c) \prod_{f \in F} P(f|c)$$

Bag-of-words document representation

I love this movie! It's sweet, but with satirical humor. The dialogue is great and the adventure scenes are fun... It manages to be whimsical and romantic while laughing at the conventions of the fairy tale genre. I would recommend it to just about anyone. I've seen it several times, and I'm always happy to see it again whenever I have a friend who hasn't seen it yet!



it	6
I	5
the	4
to	3
and	3
seen	2
yet	1
would	1
whimsical	1
times	1
sweet	1
satirical	1
adventure	1
genre	1
fairy	1
humor	1
have	1
great	1
…	…

Applying Naïve Bayes to BOW document

$$\text{positions} \leftarrow \text{all word positions in test document}$$

$$c_{NB} = \underset{c \in C}{\operatorname{argmax}} P(c) \prod_{i \in positions} P(w_i|c)$$

$$c_{NB} = \underset{c \in C}{\operatorname{argmax}} \log P(c) + \sum_{i \in positions} \log P(w_i|c)$$

Applying Naïve Bayes to BOW document

$$\text{positions} \leftarrow \text{all word positions in test document}$$

$$c_{NB} = \underset{c \in C}{\operatorname{argmax}} P(c) \prod_{i \in positions} P(w_i|c)$$

$$c_{NB} = \underset{c \in C}{\operatorname{argmax}} \log P(c) + \sum_{i \in positions} \log P(w_i|c)$$

We need to figure out what these probabilities are – this is called training!

Training Naive Bayes

- Estimating the prior probabilities: P(c)

$$\hat{P}(c) = \frac{N_c}{N_{doc}}$$

Training Naive Bayes

- Estimating the prior probabilities: P(c)

$$\hat{P}(c) = \frac{N_c}{N_{doc}}$$

- Estimating the likelihoods: P(w|c)

$$\hat{P}(w_i|c) = \frac{count(w_i, c)}{\sum_{w \in V} count(w, c)}$$

What's the problem with estimating the likelihood like this:

$$\hat{P}(w_i|c) = \frac{count(w_i, c)}{\sum_{w \in V} count(w, c)}$$

What's the problem with estimating the likelihood like this:

$$\hat{P}(w_i|c) = \frac{count(w_i, c)}{\sum_{w \in V} count(w, c)}$$

This probability could be 0 (e.g., if fantastic is in our vocab, but never occurs in positive documents):

$$\hat{P}(\text{“fantastic”}|\text{positive}) = \frac{count(\text{“fantastic”}, \text{positive})}{\sum_{w \in V} count(w, \text{positive})} = 0$$

Training Naive Bayes

- Estimating the prior probabilities: P(c)

$$\hat{P}(c) = \frac{N_c}{N_{doc}}$$

- Estimating the likelihoods: P(w|c)

First (unsuccessful) approach:

$$\hat{P}(w_i|c) = \frac{count(w_i, c)}{\sum_{w \in V} count(w, c)}$$

Training Naive Bayes

- Estimating the prior probabilities: P(c)

$$\hat{P}(c) = \frac{N_c}{N_{doc}}$$

- Estimating the likelihoods: P(w|c)

We will "smooth" the counts instead

$$\hat{P}(w_i|c) = \frac{count(w_i,c)+1}{\sum_{w\in V}(count(w,c)+1)} = \frac{count(w_i,c)+1}{\left(\sum_{w\in V} count(w,c)\right)+|V|}$$

Training Naive Bayes

- Estimating the prior probabilities: P(c)

$$\hat{P}(c) = \frac{N_c}{N_{doc}}$$

|V| size of vocabulary

- Estimating the likelihoods: P(w|c)

We will "smooth" the counts instead

$$\hat{P}(w_i|c) = \frac{count(w_i,c)+1}{\sum_{w\in V}(count(w,c)+1)} = \frac{count(w_i,c)+1}{\left(\sum_{w\in V} count(w,c)\right)+|V|}$$

Training Naive Bayes

1. Extract vocabulary, V, from the training set. Drop any unknown words in test.
2. Calculate class priors

$$\hat{P}(c) = \frac{N_c}{N_{doc}}$$

1. Calculate probability of each word in the vocabulary occurring in a document from each class type

$$\hat{P}(w_i|c) = \frac{count(w_i, c) + 1}{\sum_{w \in V} (count(w, c) + 1)} = \frac{count(w_i, c) + 1}{\left(\sum_{w \in V} count(w, c)\right) + |V|}$$

What to do with words that occur in new documents that were not in our training?

Out of Vocabulary (OOV)

- Generally just pretend they didn't happen.
- You could pretend that every unknown word occurred in the input once, but this generally doesn't help.

Situation	Can Smoothing Fix It?	Correct Action
Word is in Vocab, but count is 0 for this Class	**YES**	**Use Laplace Smoothing (+1).**
Word is NOT in Vocab (Totally new)	**NO**	**Ignore it** OR **Map to `<UNK>` token.**

OOV

	Naive Bayes (Old School)	ChatGPT (Modern LLM)
Unit	Whole Words	Subwords / Tokens
Vocabulary Size	Large (e.g., 500,000 words)	Small & Fixed (e.g., 100,000 parts)
Unknown Words	**Ignored** or <UNK>	**Broken down** into known parts
Meaning of New Words	Lost completely	Inferred from the pieces

Let's do a worked sentiment example!

	Cat	Documents
Training	-	just plain boring
	-	entirely predictable and lacks energy
	-	no surprises and very few laughs
	+	very powerful
	+	the most fun film of the summer
Test	?	predictable with no fun

A worked sentiment example with add-1 smoothing

	Cat	Documents
Training	-	just plain boring
	-	entirely predictable and lacks energy
	-	no surprises and very few laughs
	+	very powerful
	+	the most fun film of the summer
Test	?	predictable ~~with~~ no fun

1. Prior from training:

$$\hat{P}(c_j) = \frac{N_{c_j}}{N_{total}}$$

P(-) = 3/5

P(+) = 2/5

2. Drop "with"

3. Likelihoods from training:

$$p(w_i|c) = \frac{count(w_i, c) + 1}{(\sum_{w \in V} count(w, c)) + |V|}$$

$$P(\text{“predictable”}|-) = \frac{1+1}{14+20} \qquad P(\text{“predictable”}|+) = \frac{0+1}{9+20}$$

$$P(\text{“no”}|-) = \frac{1+1}{14+20} \qquad P(\text{“no”}|+) = \frac{0+1}{9+20}$$

$$P(\text{“fun”}|-) = \frac{0+1}{14+20} \qquad P(\text{“fun”}|+) = \frac{1+1}{9+20}$$

4. Scoring the test set:

$$P(-)P(S|-) = \frac{3}{5} \times \frac{2 \times 2 \times 1}{34^3} = 6.1 \times 10^{-5}$$

$$P(+)P(S|+) = \frac{2}{5} \times \frac{1 \times 1 \times 2}{29^3} = 3.2 \times 10^{-5}$$

χ^2

- χ^2 (chi-square) is a statistical test of dependence—-here, dependence between the two variables of word identity and corpus identity.

- For assessing the difference in two datasets, this test assumes a 2x2 contingency table:

	word	¬word
corpus 1	7	104023
corpus 2	104	251093

X^2

Does the word ***robot*** occur significantly more frequently in science fiction?

	robot	¬robot
sci-fi	104	1004
¬sci-fi	2	13402

= 10.3%

= 0.015%

X^2

For each cell in contingency table, sum the squared difference between observed value in cell and the expected value assuming independence.

	robot	¬robot		sum
sci-fi	104	1004		1108
¬sci-fi	2	13402		13404
sum	106	14406		

frequency
0.076
0.924

frequency	0.007	0.993	

Assuming independence:

Among 14512 words, we would expect to see 7.69 occurrences of ***robot*** in sci-fi texts.

	robot	¬robot	
sci-fi	7.69	1095.2	
¬sci-fi	93.9	13315.2	
	0.007	0.993	

0.076
0.924

χ^2

- What χ^2 is asking is: how different are the observed counts from the counts we would expect given complete independence?

	robot	¬robot	
sci-fi	104	1004	
¬sci-fi	2	13402	

	robot	¬robot	
sci-fi	7.69	1095.2	
¬sci-fi	93.9	13315.2	

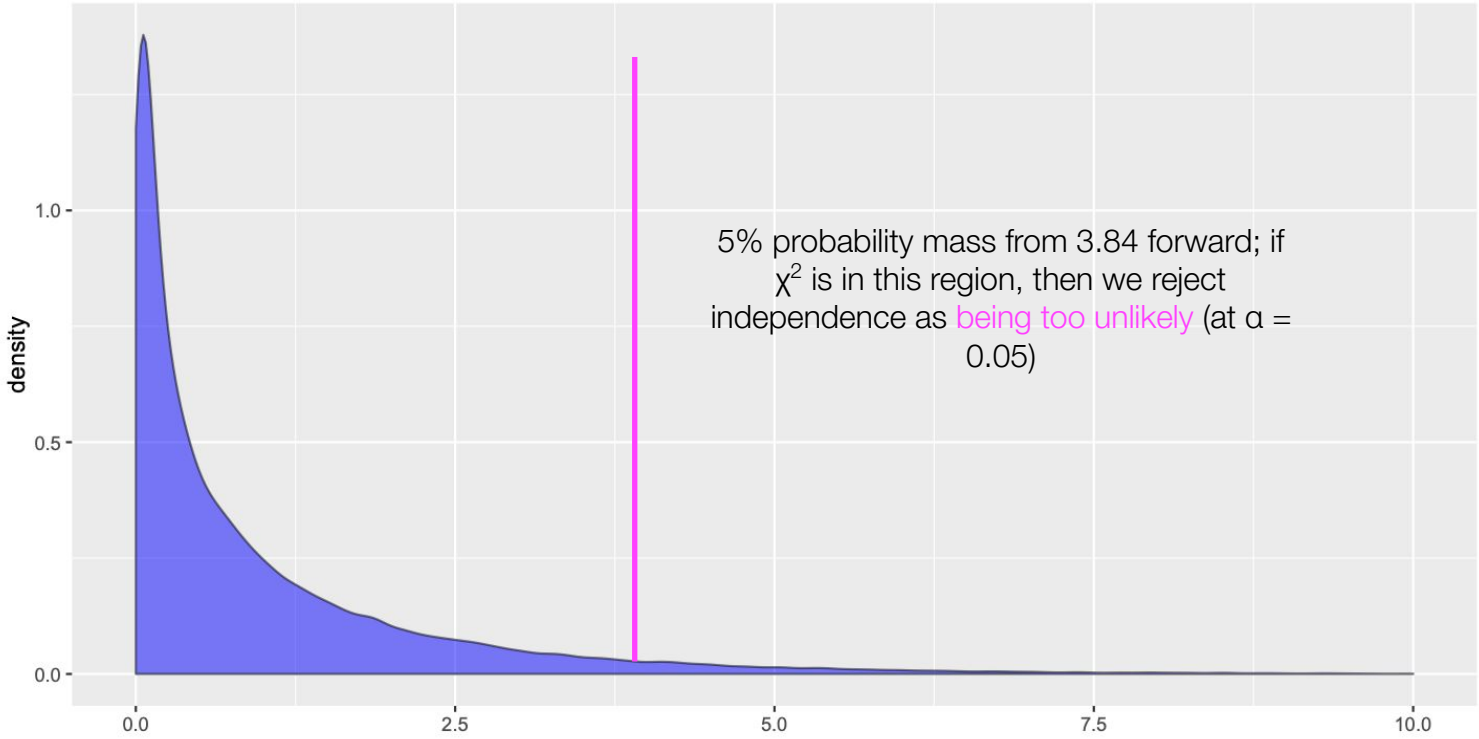
X^2

- With algebraic manipulation, simpler form for 2x2 table O (cf. Manning and Schütze 1999)

χ^2

- The χ^2 value is a statistic of dependence with a probability governed by a χ^2 distribution; if this value has low enough probability in that measure, we can reject the null hypothesis of the independence between the two variables.

X^2


density
1.0
0.5
0.0
0.0
2.5
5.0
7.5
10.0
5% probability mass from 3.84 forward; if X^2 is in this region, then we reject independence as being too unlikely (at α = 0.05)

X^2

- Chi-square is ubiquitous in corpus linguistics (and in NLP as a measure of collocations).
- A few caveats for its use:
    - Each cell should have an ***expected*** count of at least 5
    - Each observation is independent

Log-odds ratios with priors

- The odds of a word is another informative measure:

Democrat
n=1000014

Log-odds ratios with priors

- Two get a measure of difference, we can compare two odds in different corpora

- The odds ratio gives us one way of combining these into a single score

Log-odds ratios with priors

- But this is bounded by (0, ∞) and not easy to interpret with respect to the boundary (1) separating a word being likelier in corpus than another.

- We can work with the log instead, which transforms this into the space (-∞, ∞), with 0 as a boundary

Log-odds ratios with priors

- What if we have 0 counts?

- We can add pseudocounts! e.g., assume vocabulary size of 10,000 words, 100 here = 10,000 * 0.01 to account for total pseudocount mass added, and we remove 0.01 from the denominators since the denominator is the count of ¬word.

Log-odds ratios with priors

How confident are we about these estimates?

Log-odds ratios with priors

- Transform them into z-scores by dividing them by the standard deviation.

The larger the term counts (e.g., 14, 42), the more confident we can be that the difference is meaningful

Next time:

Quiz (beginning of the class)

slides credit
David Bamman (Berkeley), Alane Suhr, Kasia Hitczenko, Micha Elsner, Nathan Schneider, Sharon Goldwater, Hal Daumé III, Ellie Pavlick, Sam Bowman, Dan Jurafsky, Christopher Potts, Emily Bender