CMSC 25700/35100:
Natural Language Processing
Lecture 2: Words and Word Representations

Chenhao Tan
University of Chicago
@ChenhaoTan, chenhao@uchicago.edu

Logistics

- Quizzes (I got 60 responses, you are all quick learners!!!)
- A1 is released
- Office hours (Mourad’s office hour is 5-6 on Tuesday now)
- Coding tutorials will happen at 11am on Jan 21 and Jan 28

Basic unit of language

- Words
- But what are words and how do computers know words?

Tokenization: convert sequence of characters into sequence of tokens

“Oh!” said Lydia stoutly, “I am not afraid; for though I _am_ the youngest, I’m the tallest.”

(Austen, 1813)

“Oh!” said Lydia stoutly, “I am not afraid; for though I _am_ the youngest, I’m the tallest.”

(Austen, 1813)

“Oh!”	not	the
said	afraid;	youngest,
Lydia	for	I’m
stoutly,	though	the
“I	I	tallest.”
am	_am_	

“Oh!” said Lydia stoutly, “I am not afraid; for though I _am_ the youngest, I’m the tallest.”

(Austen, 1813)

“ Oh ! ” said Lydia stoutly , “ I am not afraid ; for though I _ am _ the youngest , I ’m the tallest . ”

- Most tokenizers are rule-based
- several conventions:

	Penn Treebank	Moses
don’t	do n’t	don ’t
aren’t	are n’t	aren ’t
can’t	ca n’t	can ’t
won’t	wo n’t	won ’t

- important to be consistent across NLP systems, match tokenization of external tools/resources
- see `nltk.tokenize` (also for sentence tokenization)

- Chinese, Japanese, Thai: no spaces between words
- Tokenization becomes highly non-trivial!

姚明进入总决赛
“Yao Ming reaches the finals”

- Multiple conventions:

姚明 进入 总决赛
“YaoMing reaches finals”

Chinese Treebank

姚 明 进入 总 决赛
“Yao Ming reaches overall finals”

Peking University

Example from Chen et al. (2017), cited in Jurafsky & Martin (SLP3)

“ oh ! ” said lydia stoutly , “ i am not afraid ; for though i _ am _ the youngest , i ’m the tallest . ”

3	i	1	!	1	oh
2	,	1	.	1	said
2	_	1	;	1	stoutly
2	am	1	afraid	1	tallest
2	the	1	for	1	though
2	“	1	lydia	1	youngest
2	”	1	not	1	’m

```
3 i          1 !          1 oh
2 ,          1 .          1 said
2 _          1 ;          1 stoutly
2 am         1 afraid     1 tallest
2 the        1 for        1 though
2 “          1 lydia      1 youngest
2 ”          1 not        1 ’m
```

type: a unique word (an entry in a vocabulary or dictionary)

token: an instance of a type in a corpus

3	i	1	!	1	oh
2	,	1	.	1	said
2	_	1	;	1	stoutly
2	am	1	afraid	1	tallest
2	the	1	for	1	though
2	“	1	lydia	1	youngest
2	”	1	not	1	’m

two types of counts:

type count =

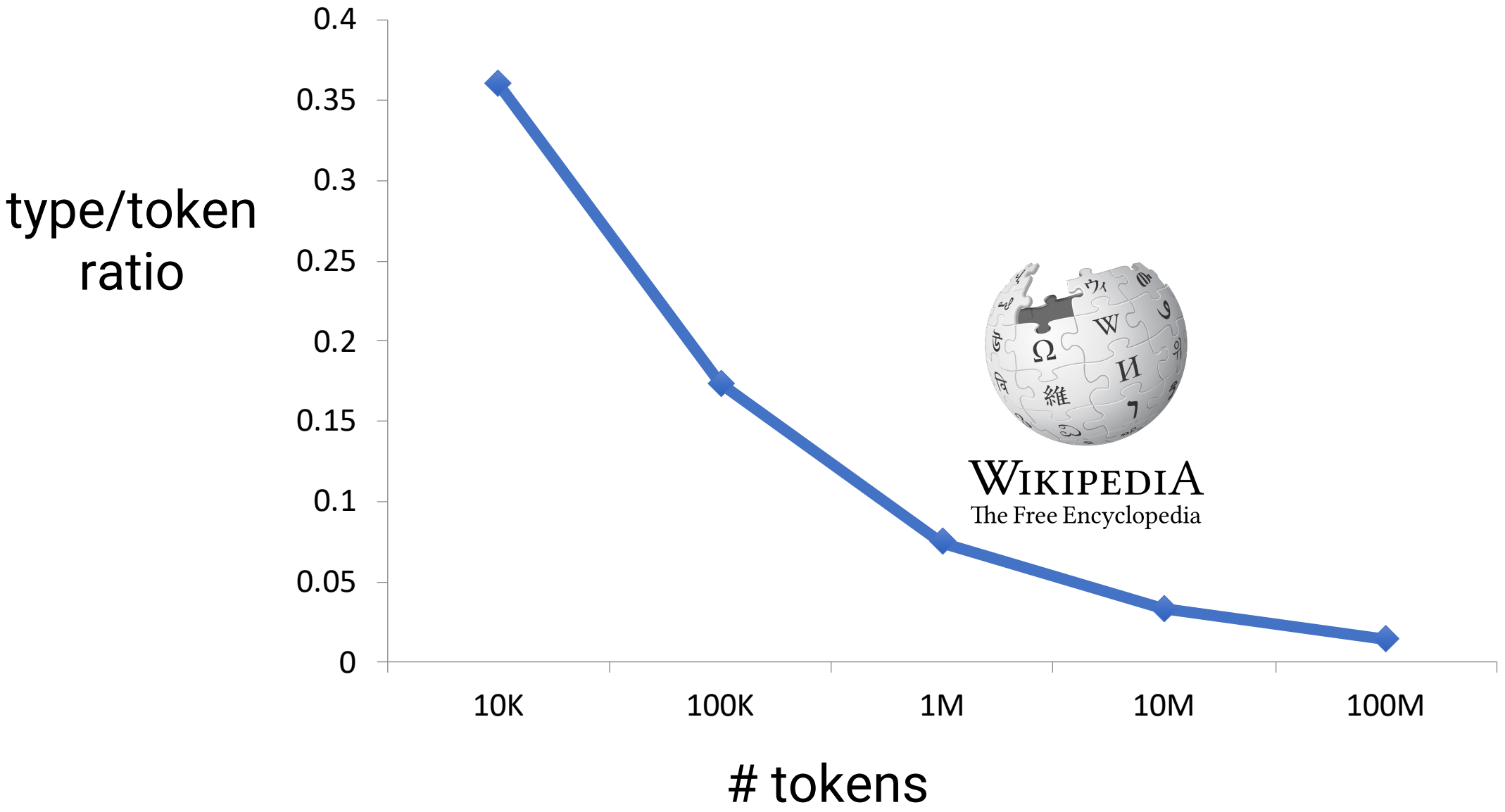
token count =

useful statistic: **type/token ratio**

(here, 21/29 = 0.724)

How does the type/token ratio change when adding more data?

more data → lower type/token ratio

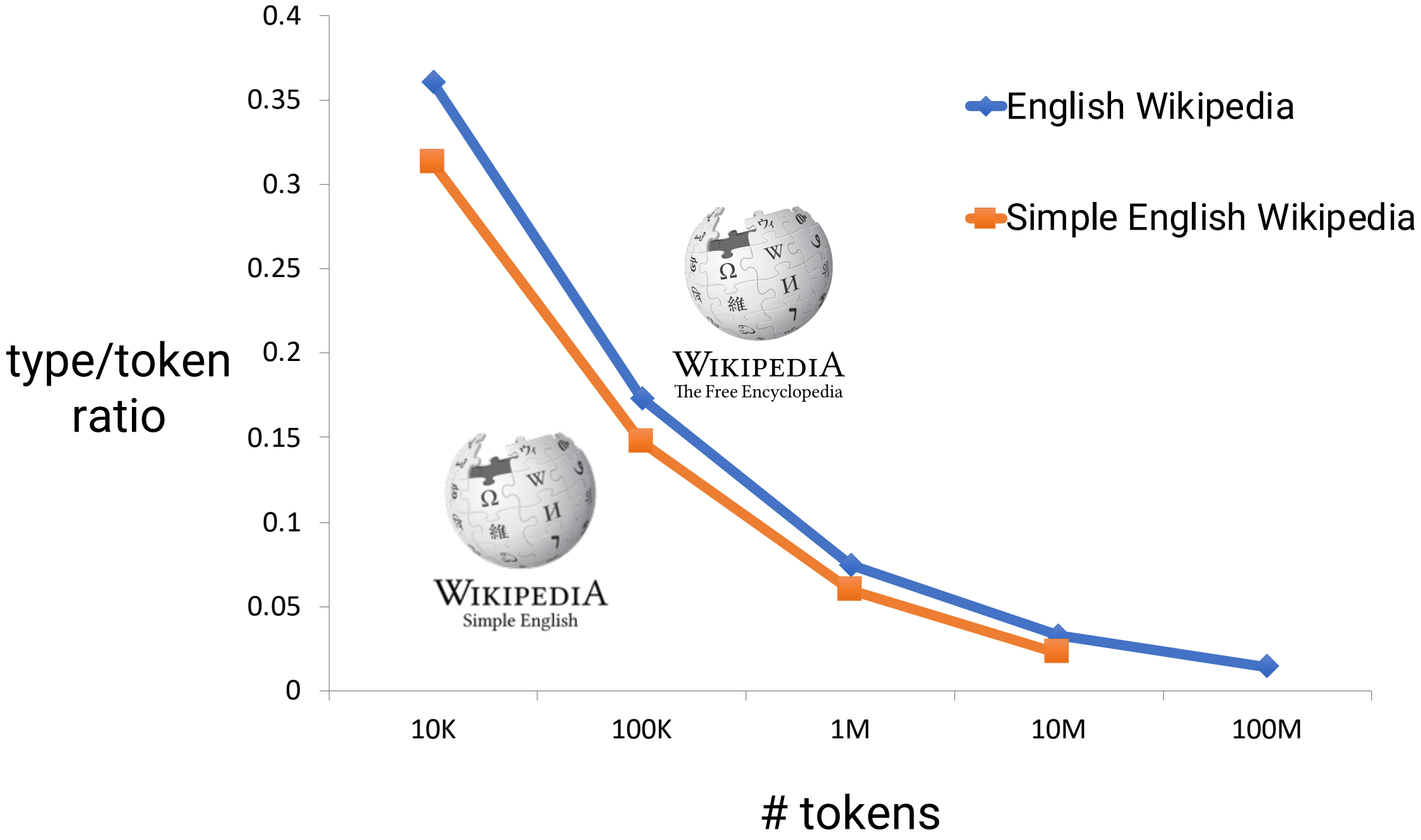


WIKIPEDIA
The Free Encyclopedia

vs.

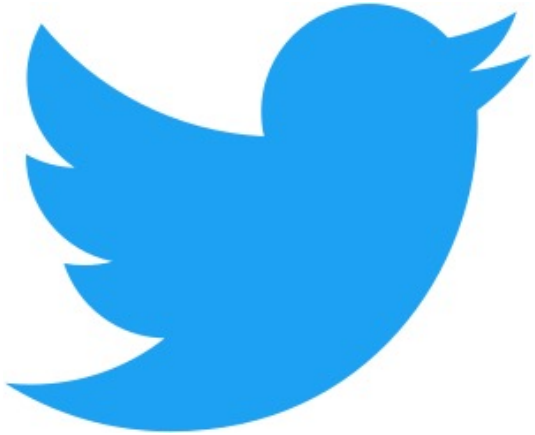
WIKIPEDIA
Simple English

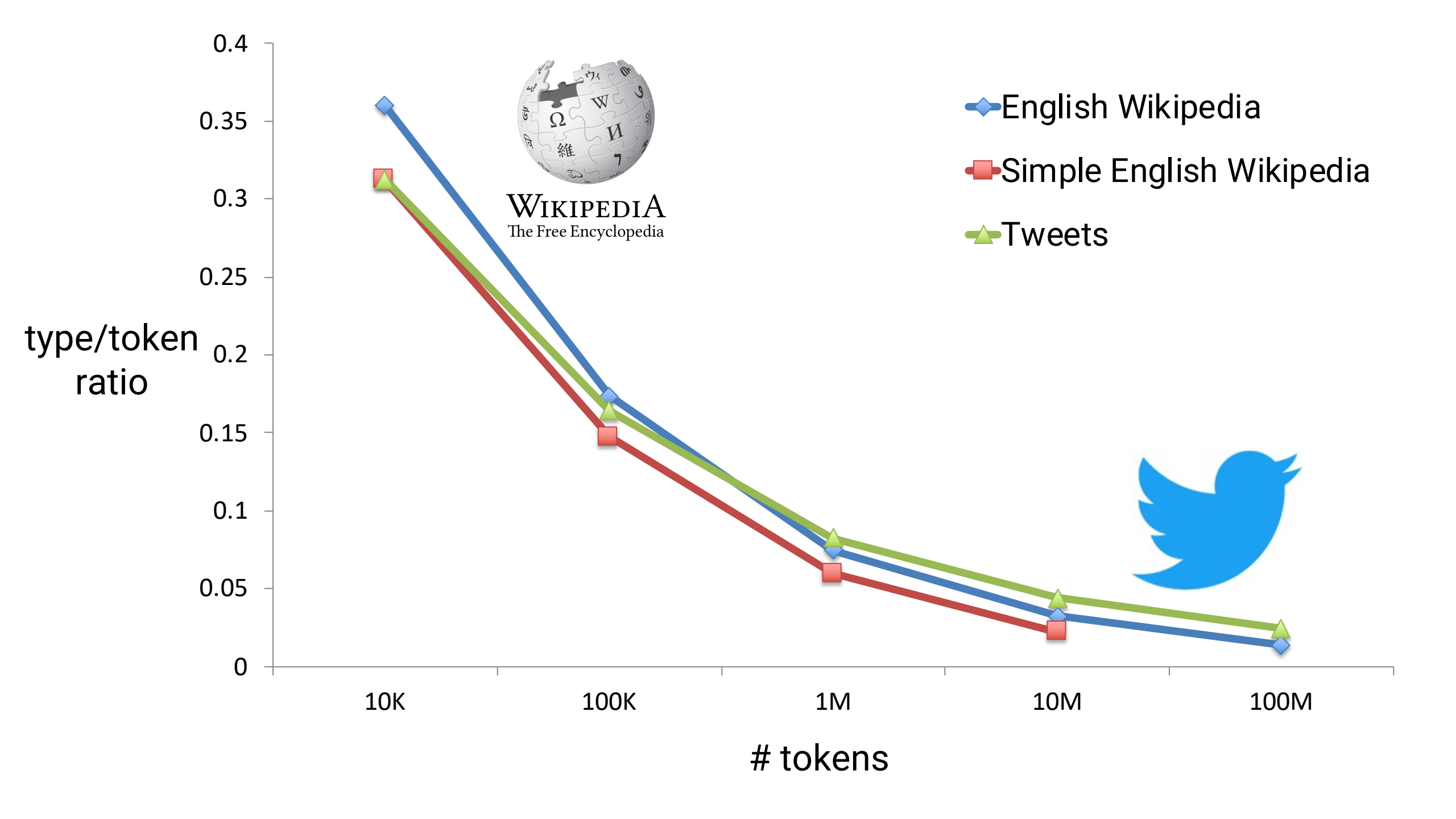
Which has a higher type/token ratio?


type/token ratio
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
10K
100K
1M
10M
100M
tokens
English Wikipedia
Simple English Wikipedia
WIKIPEDIA
The Free Encyclopedia
WIKIPEDIA
Simple English

WIKIPEDIA
The Free Encyclopedia

vs.




0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
type/token ratio
WIKIPEDIA
The Free Encyclopedia
English Wikipedia
Simple English Wikipedia
Tweets
10K
100K
1M
10M
100M
tokens

```
224571 really
  1189 rly
  1119 realy
   731 rlly
   590 reallly
   234 realllly
   216 reallyy
   156 relly
   146 reallllly
   132 rily
   104 reallyyy
    89 realllllly
    89 reeeally
    84 reaaally
    82 reaally
    72 reeeeally
    65 reaaaally
    57 reallyyyy
    53 rilly
    50 reallllllly
    48 reeeeeally
    41 reeally
    38 really2
    37 reaaaaally
    35 reallyyyyy
    31 reely
    30 reallyyy
    27 reaaly
    27 reallyy
    26 reallyyyy
    25 realllllllly
    22 reaaallly
    21 really-
    19 reeaally
    18 reallllyyy
    16 reaaaallly
    15 reaallly
    15 realllllllly
    15 reallllyy
    15 reallyreally
    15 realyy
    14 reallllyyyy
    14 reeeeeeally
    13 reeeaaally
    12 reaaaaaally
    12 rreally
    11 reaallyy
    11 reallllllyyy
    11 reeeallly
    11 reeeeallly
    10 reaaaly
    10 reallyreallyreally
     9 r)eally
     9 really-really
     9 reallys
     9 reeeeeeeally
     8 realky
     8 reallyyyyyy
     8 reallyyyyyyy
     8 reeeaally
     7 r3ally
     7 raelly
     7 reaaaaaaally
     7 reallllllllllly
     7 reallllllyyy
     7 reeeeaaally
```

7 reeeealy
7 reeeeeeeeeally
7 relaly
6 r-e-a-l-l-y
6 r-really
6 reaaaaaallly
6 reallllllllly
6 realllyyyyy
6 realyl
6 reeeaaaally
6 reeeaaallly
6 reeeaaalllyyy
5 reaaaaallly
5 reaaaalllly
5 reaalllyy
5 reallllllllllly
5 reallllllllllly
5 reeallyyy
5 reeeeaaallly
5 reeeeaally
5 reeeeeeeeally
5 rellly
5 rrly
5 rrrreally
4 reaaaaly
4 reaaalllly
4 reaaalllyy
4 reaalllly
4 reaalllyyy
4 reallllllyyyy
4 reallllyyyy
4 reeaaaally
4 reeealy
4 reeeeeeeeeally
4 rllly
3 r34lly
3 r]eally
3 reaaaaaaaally
3 reaaaaaly
3 reaaaalllllly
3 reaaaallyy
3 reaaallyy
3 reaalllllly
3 reaallyyyy
3 realiy
3 reallllllllllllllly
3 reallllllyy
3 reallllllyyyy
3 reallllllyyyyyyy
3 reallllyyyyy
3 realluy
3 really)
3 reallyl
3 reallyyyyyyyyy
3 reeaaallly
3 reeaalllly
3 reeaalllyyy
3 reeaaly
3 reeallly
3 reealy
3 reeeaaallllyyy
3 reeeaallly
3 reeeeaaaaally
3 reeeeallllly
3 reeeeeallllly
2 reaaaaaaaaaally

2 reaaaaaaaaally
2 reaaaaaaaallly
2 reaaaaaallllllyyy
2 reaaaaalllllly
2 reaaaaalllllly
2 reaaallllllyyy
2 reaaallllyyy
2 reaaalllyyy
2 reaallllyy
2 reaallllyyy
2 reaallyyy
2 reaalyy
2 reallllllllllllllly
2 reallllllllllllly
2 reallllllyy
2 reallllllyyyyy
2 reallllllyyyyy
2 reallllyy
2 reallllyyyyyy
2 reallllyyyyyy
2 reallyyyyyy
2 really*
2 really/
2 reallyyyyyyyy
2 reallyyyyyyyyyyy
2 realyyy
2 reaqlly
2 reeaaally
2 reeaallly
2 reeaalllyy
2 reeaallyy
2 reeallyy
2 reeeallyy
2 reeeeaaaalllyyy
2 reeeeaaaally
2 reeeeaaallly
2 reeeeaaallllyyyy
2 reeeeallllyyy
2 reeeeallyyy
2 reeeeeaaalllllly
2 reeeeeaaally
2 reeeeeaally
2 reeeeeallly
2 reeeeeealy
2 reeely
2 rellys
2 rellyy
2 reqally
2 rlyyy
2 rlyyyy
2 rreeeallyy
2 rrreally
1 r-r-r-really
1 r3aly
1 r3ly
1 raaahhhlllaaayyyy
1 raeally
1 re-e-e-eally
1 re-eaaaaaaaly
1 re-he-he-he-ealy
1 re-he-he-heeeeally
1 re3ally
1 rea(l)ly
1 reaaaaaaaaaaaaaaaaaaally
1 reaaaaaaaaaaaaaaaally
1 reaaaaaaaaaaaaaaaallllly

1 reaaaaaaaaaaaaaally
1 reaaaaaaaaaaaally
1 reaaaaaaaaaaalllllly
1 reaaaaaaaaaaallllly
1 reaaaaaaaaaaally
1 reaaaaaaaaaalllllllyyyyyyyyy
1 reaaaaaaaaallly
1 reaaaaaaaallllllyyyyy
1 reaaaaaaaallllly
1 reaaaaaaaallllyyyy
1 reaaaaaallllly
1 reaaaaaalllyyyy
1 reaaaaalllllllyyyy
1 reaaaaalllllyyy
1 reaaaaalllllyyyy
1 reaaaaalllllyyyyy
1 reaaaaalllllyyyyyyy
1 reaaaaallllyy
1 reaaaaallllyyyyy
1 reaaaaallyy
1 reaaaallllllly
1 reaaaallllllyy
1 reaaaalllllyyy
1 reaaaalllllyyyyy
1 reaaaallllyyy
1 reaaaallllyyyyy
1 reaaaalllyy
1 reaaaalllyyy
1 reaaaalllyyyy
1 reaaaallyyy
1 reaaalllllllly
1 reaaalllllllyyy
1 reaaallllly
1 reaaalllllyyyy
1 reaaalllllyyyyy
1 reaaalllyyyyy
1 reaaallyyy
1 reaaallyyyy
1 reaalllllly
1 reaalllllyyyy
1 reaallllyyyy
1 reaalllyyyy
1 reaalllyyyyy
1 reaalllyyyyyy
1 real(ly
1 realaaay
1 realkly
1 reall(y
1 reallhy
1 reallllllllllllllllllllllllllllllllll
llllllllllllllllllllllllllllllllllly
1 reallllllllllllllllllllllllllllllllll
llllllllllllllllllllly
1 reallllllllllllllllllllllllllllllllll
llllllllllly
1 realllllllllllllllllllllllllllllllllly
1 reallllllllllllllllllllllllllllly
1 realllllllllllllllllllllllllllly
1 realllllllllllllllllllllllyyyyy
1 realllllllllllllllllllllly
1 reallllllllllllllllllllly
1 reallllllllllllllyyyyyy
1 reallllllllllyyyyy
1 realllllllllyyy
1 realllllllllyyyy
1 realllllllllyyyyy
1 realllllllllyyyyyy
1 reallllllllyyyy
1 reallllllllyyyyyyy
1 reallllllllyz
1 realllllyly
1 realllllyyyyy
1 reallllyyyyyyy
1 reallllyyyyyyyyy
1 realllylyyy
1 realllyyyyyyy
1 realllyyyyyyyyy
1 really(really
1 really//
1 really/really/really
1 really100
1 really2x
1 really:')
1 really^^
1 really_
1 reallyii
1 reallyreallyreallyreallyreallyr33lly
1 reallyreallyreallyreallyreallyreally
reallyreallyreallyreallyreallyreally
reallyreallyreallyreallyreally

1 reallyyyyyyyyyyy
1 really😒
1 realoly
1 realys
1 realyyyyy
1 real•ly
1 reawly
1 ree-hee-heally
1 reeaaaaaaaaaaaaaaaaalllllllllyyy
1 reeaaaaaaaallllllllly
1 reeaaaaaallllly
1 reeaaaallly
1 reeaaalllly
1 reeaaallllyyy
1 reeaaallyyy
1 reeaaaly
1 reeaalllllyyy
1 reeaallllyyy
1 reeaallyyy
1 reealllly
1 reealllyyy
1 reealllyyyyy
1 reeeaaaaaaaly
1 reeeaaaaally
1 reeeaaaalllllly
1 reeeaaaallllyyy
1 reeeaaaallllyyyy
1 reeeaaaallly
1 reeeaaaalllyyy
1 reeeaaaaly
1 reeeaaalllllyyyy
1 reeeaaallllyyyy
1 reeeaaallyy
1 reeeaaallyyy
1 reeeaaaly
1 reeeaalllly
1 reeeaallyy
1 reeeaallyyy
1 reeeallllly
1 reeealllly
1 reeeallllyy
1 reeeallllyyy
1 reeeallllyyyy
1 reeealllyyy
1 reeealllyyyy
1 reeeallys
1 reeeeaaaaaaallllllyyyyyyy
1 reeeeaaaaaallllllyyyyy
1 reeeeaaaaalllllllly
1 reeeeaaaaallllly
1 reeeeaaaaallly
1 reeeeaaaaalllyyy
1 reeeeaaaalllly
1 reeeeaaaallllyyyyyy
1 reeeeaaallllyy
1 reeeeaaalllyy
1 reeeeaaallyyy
1 reeeeaaaly
1 reeeeaallly
1 reeeeaalllyy
1 reeeeaaly
1 reeeealllllly
1 reeeeallllyy
1 reeeealllyy
1 reeeealllyyy
1 reeeealllyyyy
1 reeeeeaaaallllllllly
1 reeeeeaaaaally
1 reeeeeaallllly
1 reeeeeaalllllyyy
1 reeeeeaalllly
1 reeeeeallllyyy
1 reeeeealy
1 reeeeeeaaaaaally
1 reeeeeeaaaaalllllyyyy
1 reeeeeeaaaaally
1 reeeeeeaaaalllly
1 reeeeeeaally
1 reeeeeeaaly
1 reeeeeeallllly
1 reeeeeealllyyyy
1 reeeeeeeaaalllly
1 reeeeeeeaally
1 reeeeeeeallyy
1 reeeeeeealy
1 reeeeeeeeaaaaaaaalllllyyyyyy
1 reeeeeeeeeaaaaaaaaalllllllllyyyyyyyy
1 reeeeeeeeeaaaaaaallllllllyyyyyyyy
1 reeeeeeeeeaaaaaalllyyy
1 reeeeeeeeeaaally
1 reeeeeeeeeaally
1 reeeeeeeeeallllly
1 reeeeeeeeeeaaally

1 reeeeeeeeeeaally
1 reeeeeeeeeeeally
1 reeeeeeeeeeeeally
1 reeeeeeeeeeeeeaaally
1 reeeeeeeeeeeeeeeaaaallly
1 reeeeeeeeeeeeeeeeeally
1 reeeeeeeeeeeeeeeeeeeeeesalllllllllllllllllllllllly
1 reeeeeeelly
1 reeeeeely
1 reeeeelly
1 reeeeely
1 reeeellllyyy
1 reeellllyy
1 reellly
1 reelllyy
1 reheheally
1 relally
1 rellllllly
1 rellly
1 rellyrell
1 rellzy
1 rieeely
1 rlllllly
1 rllllllyy
1 rllly
1 rllyrlly
1 rllyy
1 rllyyy
1 rlyrlyrly
1 rlyy
1 rraarreellyy
1 rreaalllyyy
1 rreaally
1 rreeaalllyyyy
1 rreeaallllyy
1 rreeaallyyy
1 rreealy
1 rreeeaaaaallllyyyy
1 rreeeaaallllyyy
1 rreeeeeeaaaaallllllllyyyyyy
1 rreeeeeeally
1 rreeeeeely
1 rrreallyyy
1 rrreeeaaalllyyy
1 rrreeealllyyy
1 rrreeeeaallllly
1 rrreeeeally
1 rrrlyyy
1 rrrreeeally
1 rrrreeeeeeaaaaallllllllyyyyyy
1 rrrrreeeeaaallly
1 rrrrreeeeaaalllyy
1 rrrrrreally
1 rrrrrrealy
1 rrrrrrreeeeeeaaaallllyyyyyyy
1 rrrrrrrrreally
1 rrrrrrrrrrrrrrreeeeeeeeeeeaaaaaaallllllyyyyyy

How many words are there?

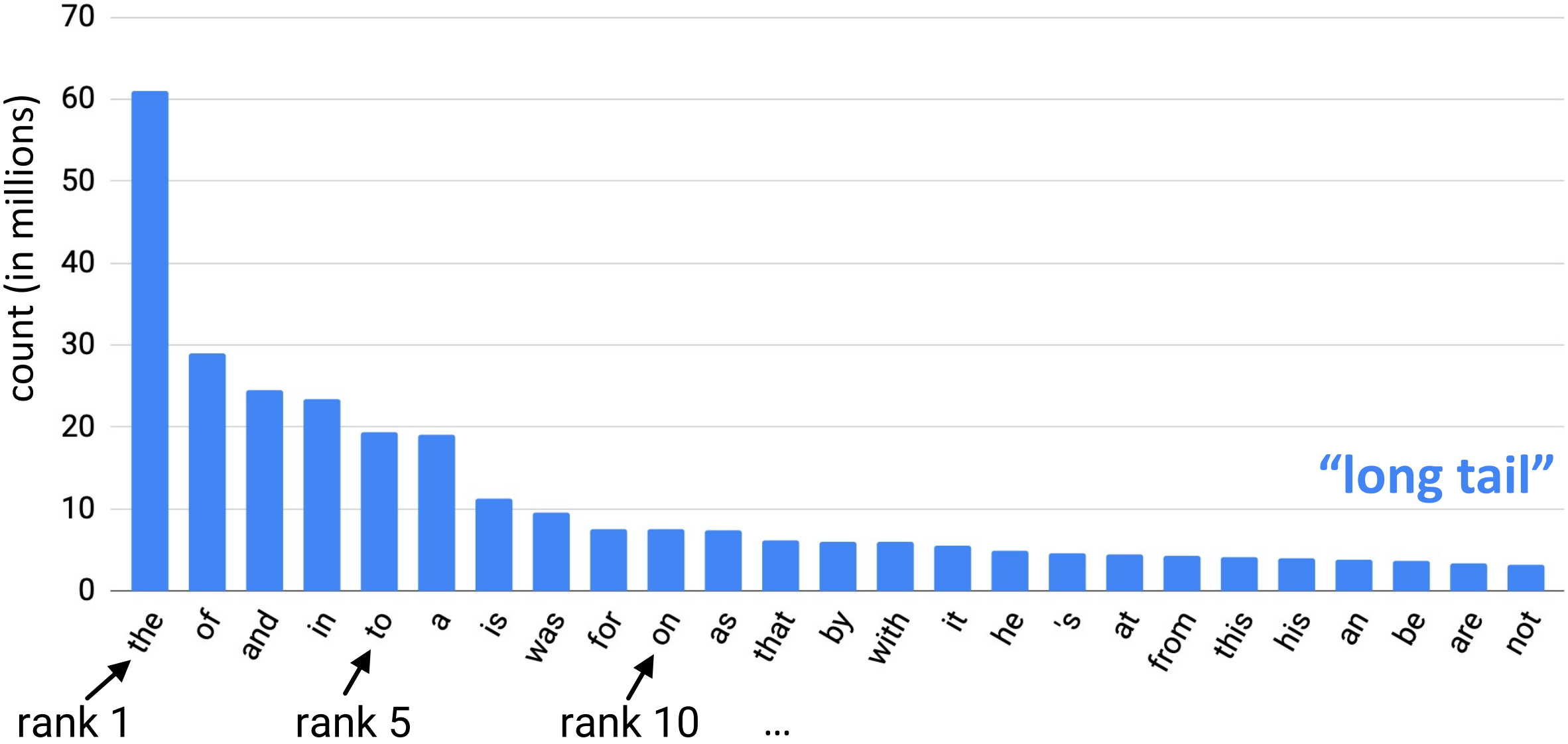
- size of vocabulary continues to grow as you collect more data

- you'll never find all the words

How are words distributed?

224571	really	1	rreeeeeeaaaaallllllllyyyyyy
1189	rly	1	rreeeeeeallly
1119	realy	1	rreeeeeely
731	rlly	1	rrreallyyy
590	reallly	1	rrreeeaaalllyyy
234	realllly	1	rrreeealllyyy
216	reallyy	1	rrreeeeaallllyy
156	relly	1	rrreeeeallly
146	reallllly	1	rrrlyyy
132	rily	1	rrrreeeally
104	reallyyy	1	rrrreeeeeeaaaaallllllllyyyyyy
89	realllllly	1	rrrrreeeeaaallly
89	reeeally	1	rrrrreeeeaaalllyy
84	reaaally	1	rrrrrreally
82	reaally	1	rrrrrrealy
72	reeeeally	1	rrrrrrreeeeeeaaaallllyyyyyyy
65	reaaaally	1	rrrrrrrrrreally
…		1	rrrrrrrrrrrrrrreeeeeeeeeeeeaaaaaaallllllyyyyyy

Zipf's law: frequency of a word is inversely proportional to its rank in the word frequency list



The Long Tail

- there are so many word types!
- but words have **internal structure**

Morphology in NLP

- NLP problems that address morphology:

  lemmatization

  stemming

Terminology

- **lemma**
  - canonical/dictionary form of a word
  - words with same lemma have same stem, part of speech, rough semantics
- **word form**
  - full inflected or derived form of a word as it appears in text

word form	lemma
run	run
ran	run
running	run

Sec. 2.2 (J&M)

Lemmatization

- **lemmatization**: convert wordform to lemma

```
am, is, are → be
car, cars, car's, cars' → car

the boy's cars are different colors
                              ↓
the boy car be different color
```

- mostly about finding the correct dictionary entry, but this may depend on the context

Sec. 2.4.4 (J&M)

Stemming

- **stemming**: reduce words to their stems by removing affixes
  - usually implemented with language-specific, manually-designed rules
  - commonly used in information retrieval
  - example:

```
Caillou is an average, imaginative four-year-old boy with a love for
forms of transportive machinery such as rocket ships and airplanes.
```

```
Caillou is an averag imagin four year old boi with a love for
form of transport machineri such as rocket ship and airplan
```

Sec. 2.4.4 (J&M); en.wikipedia.org/wiki/Caillou

Porter's Algorithm
(the most common English stemmer)

Step 1a

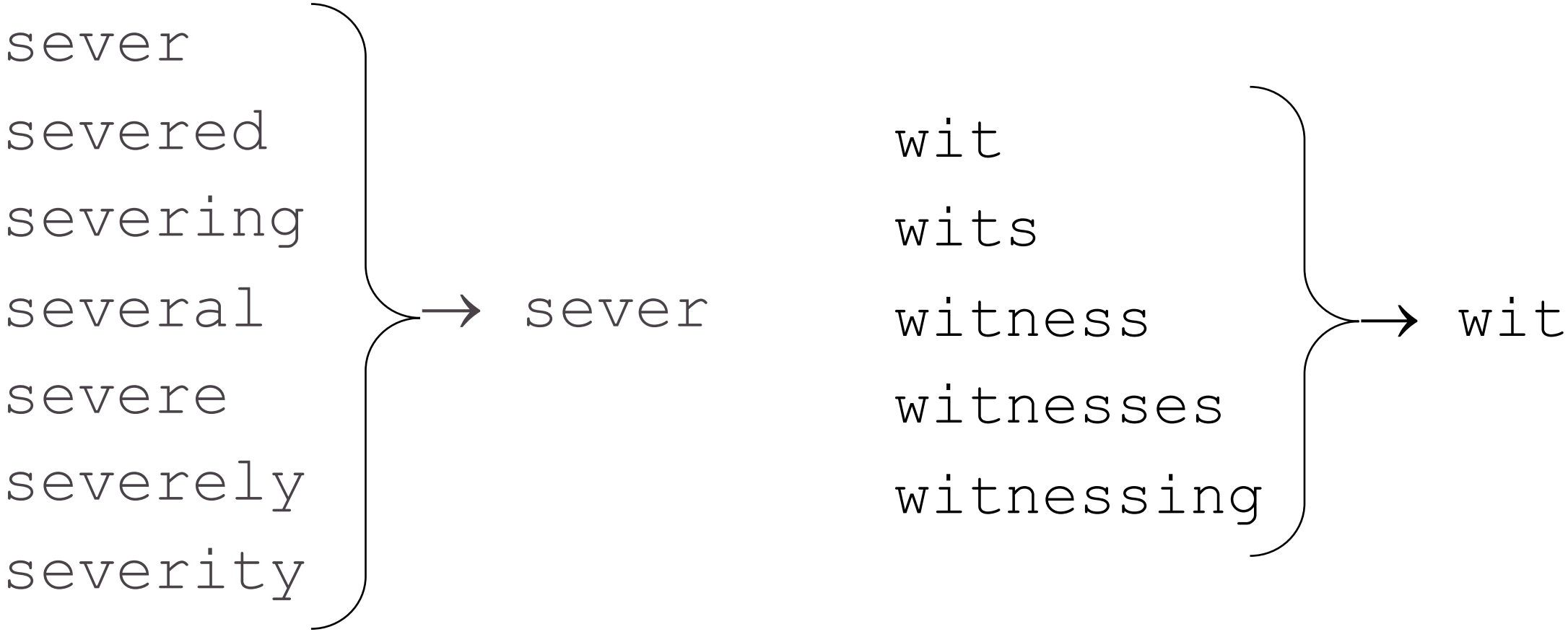
```
sses → ss  caresses → caress
ies  → i   ponies   → poni
ss   → ss  caress   → caress
s    → ø    cats      → cat
```

Step 1b

```
(*v*)ing → ø  walking   → walk
               sing      → sing
(*v*)ed  → ø  plastered → plaster
…
```

Sec. 2.4.4 (J&M)

Idiosyncrasies of the Porter Stemmer



Lemmatization vs. Stemming

- Lemmatization
  - viewed as an NLP task
  - solved with dictionary look-up, possibly with machine learning
- Stemming uses manually-defined rules
  - simple and fast, but limited due to reliance on rules
  - may conflate words erroneously
- Both may remove information
  - To mitigate, combine lemma/stem form with original form, use both!

Data-Driven Segmenters

- Segment words into pieces (subword units or wordpieces) based on common character sequences in a dataset
- Most popular methods:
  - Byte pair encoding (BPE)
  - SentencePiece's unigram language model (LM)
- These are efficient and effective, but they don't necessarily correspond to morphology; splits may be arbitrary
- Very popular when using neural networks (BERT, GPT, etc)

Sec. 2.4.3 (J&M)

Byte Pair Encoding (BPE)
(Gage, 1994)

- Simple data compression technique
- Iteratively replaces most frequent pair of bytes in a sequence with a single, unused byte
- Sennrich et al. (2016) adapted BPE for segmenting words

Byte Pair Encoding for Words

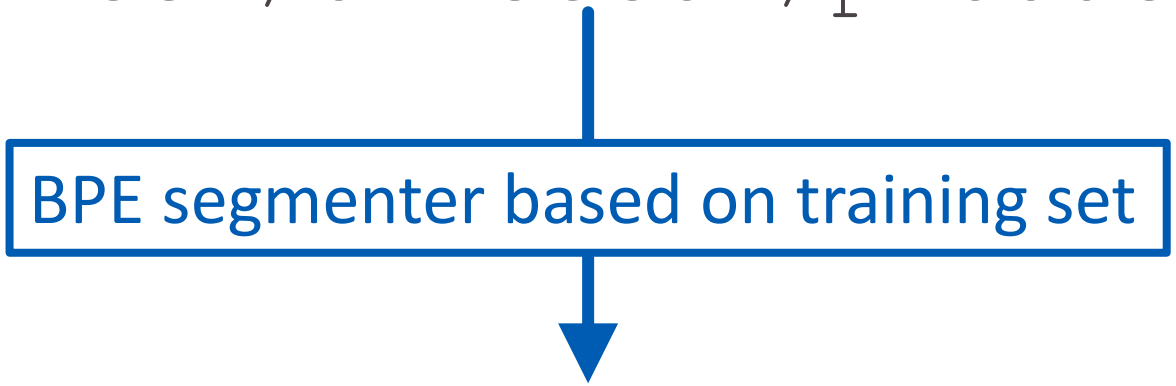
- "merge": operation that combines two consecutive units into a single unit
  - initially, units are characters (e.g., `s` or `t`)
  - after merges, units become character sequences (e.g., `st` or `books`)
- greedy algorithm:
  - merge 2 units with the largest 2-unit sequence count, produce merged unit
  - replace all instances of that 2-unit sequence with the merged unit, recompute counts

Sennrich et al. (2016): *Neural Machine Translation of Rare Words with Subword Units*

Example from movie review dataset (Stanford Sentiment Treebank):

word that was not in training set:

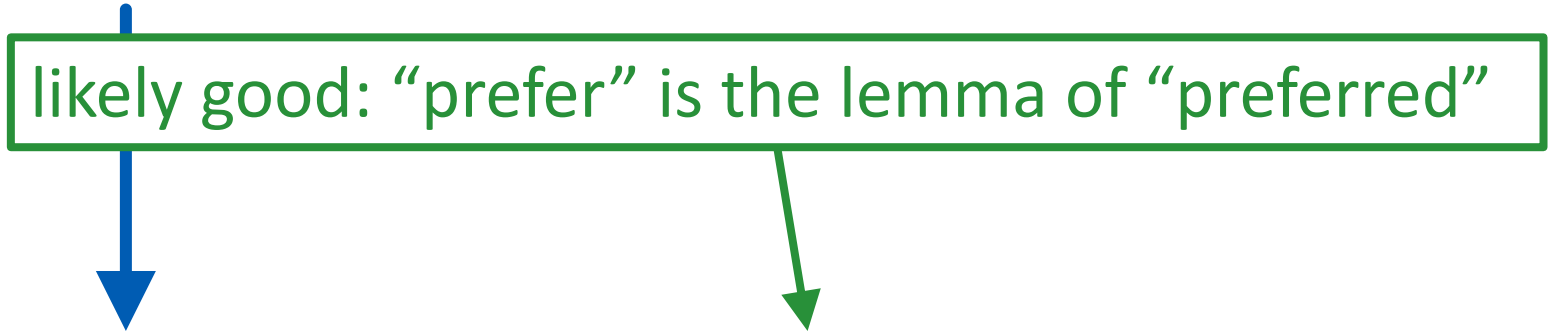
`writer/director/producer`



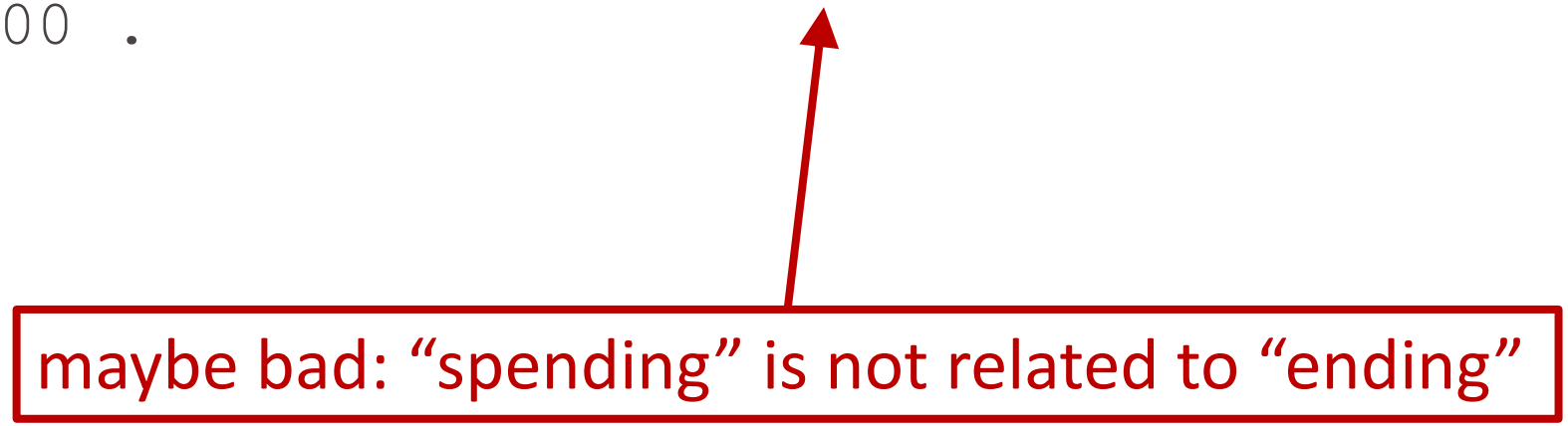
`writ@@ er@@ /@@ direct@@ or@@ /@@ producer`

(recover original text by removing "`@@ `")

```
It wouldn't be my preferred way of spending 100
minutes or $ 7.00 .
```




```
It wouldn't be my prefer@@ red way of sp@@ ending 100
minutes or $ 7@@ .@@ 00 .
```



Summary (1/3)

- to do NLP on some text, we need to preprocess it:
  - tokenize documents into sentences
  - tokenize sentences into tokens
- rule-based tokenizers exist for many languages
- for writing systems without whitespace, tokenization becomes complex

Summary (2/3)

- useful terms: **type**, **token**, **type/token ratio**
- when adding data, number of types keeps increasing
- most types are extremely rare (Zipf's law)
- people can often understand a novel word type based on its internal structure and the context of its use

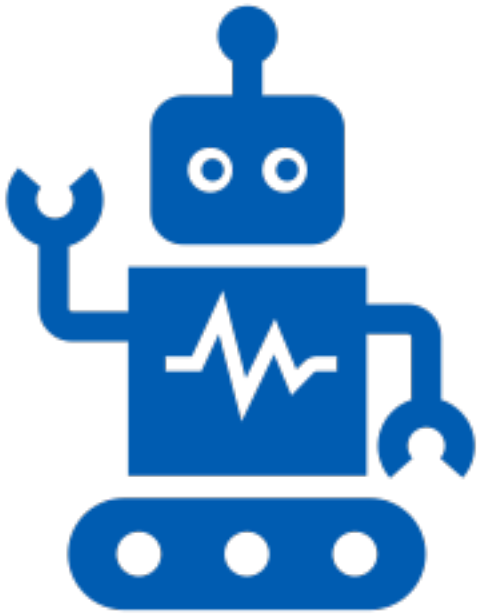
Summary (3/3)

- **morphology**: study of how words are built from morphemes
- **lemmatization**: convert wordform to lemma (may depend on context)
- **stemming**: removing affixes from words to get stems (simple, rule-based)
- data-driven segmentation (BPE, unigram LM) splits words based on data, very common in deep learning

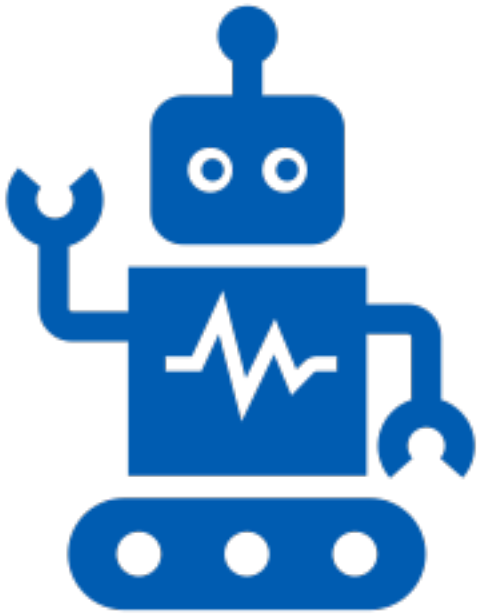
How do represent words computationally?

荃者所以在鱼，得鱼而忘荃　Nets are for fish;
Once you get the fish, you can forget the net.
言者所以在意，得意而忘言　Words are for meaning;
Once you get the meaning, you can forget the words

庄子(Zhuangzi), Chapter 26



cat	chef	chicken	civic	cooked	council	…
↓	↓	↓	↓	↓	↓	
17	91	253	104	5	6001	…



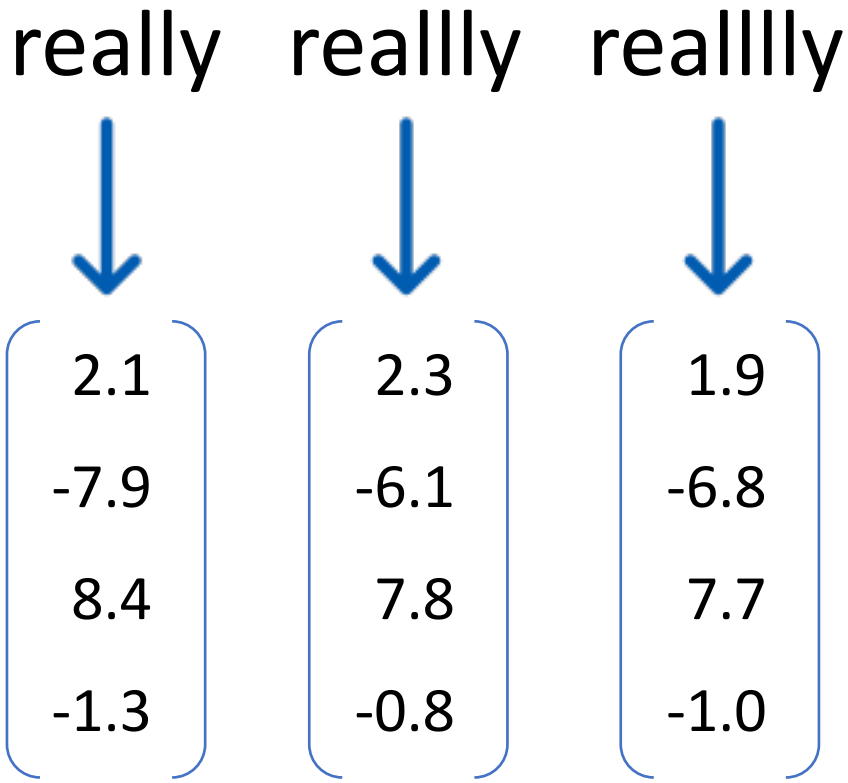
cat	chef	chicken	civic	cooked	council	…
0.1	-0.1	-0.4	0.1	-0.5	0.6	
7.9	2.1	2.4	0	-1.1	-1.3	
2.4	3.8	9.7	-1.5	7.6	0	
-1.3	-0.1	-1.0	2.4	-3.1	3.4	
0.5	5.3	3.2	0.2	4.2	-0.6	

“embeddings”

Motivation for word embeddings?

Variability

multiple forms, similar meaning



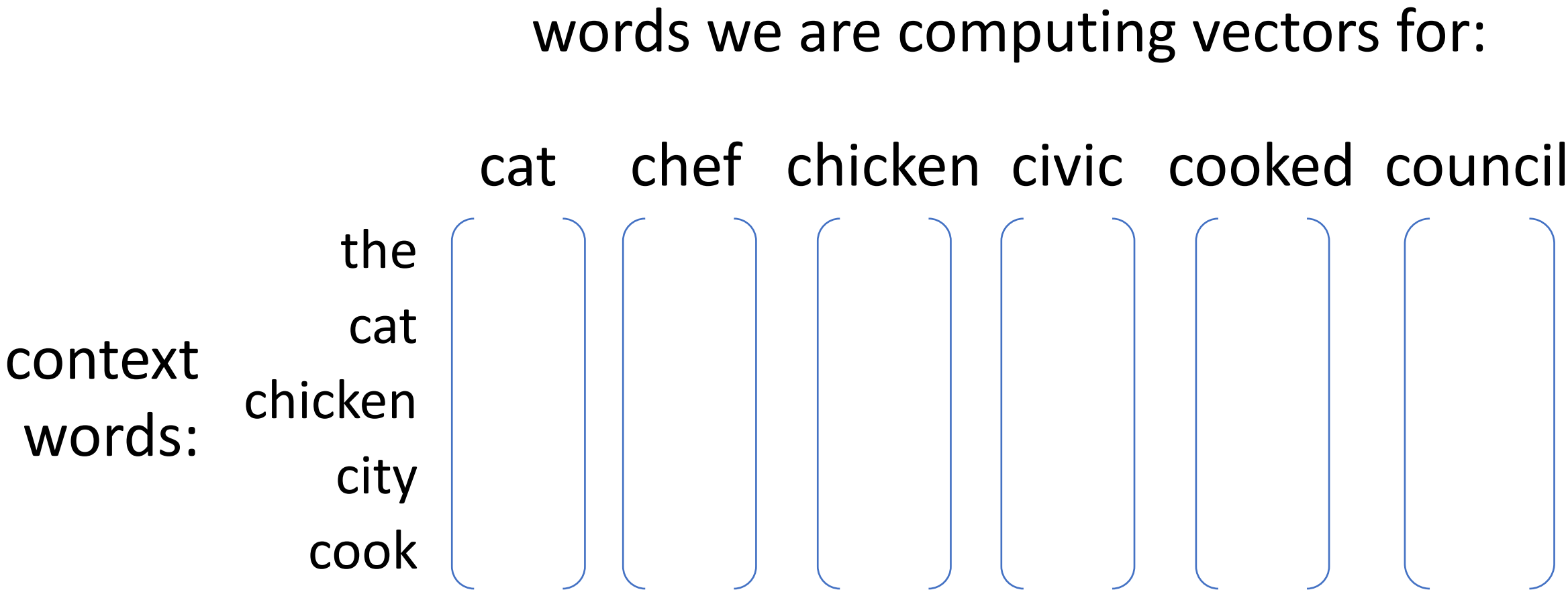
How should we embed words?

distributional hypothesis: words that appear in similar contexts have similar meanings (Joos, 1950; Harris, 1954; Firth, 1957)

"You shall know a word by the company it keeps."

–Firth (1957)

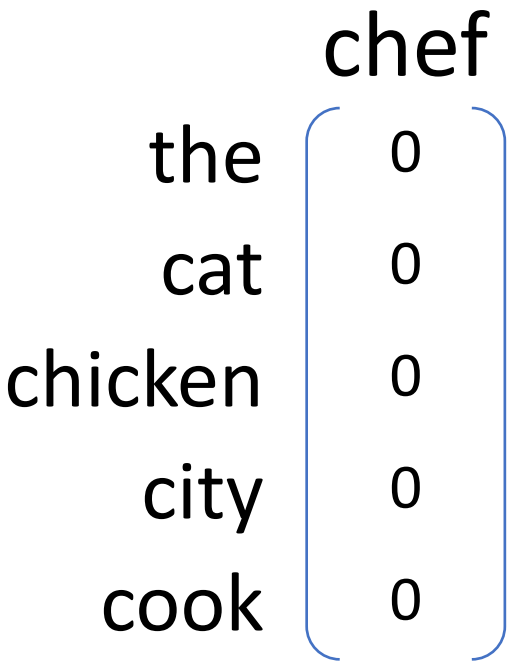
Ch. 6 (J&M)


words we are computing vectors for:
cat
chef
chicken
civic
cooked
council
context words:
the
cat
chicken
city
cook

… , the club may also employ a **chef** to prepare and cook food items .

… is up to remy , linguini , and the **chef** colette to cook for many people …

… cooking program the cook and the **chef** with simon bryant , who is …



… , the club may also employ a **chef** to prepare and cook food items .

… is up to remy , linguini , and the **chef** colette to cook for many people …

… cooking program the cook and the **chef** with simon bryant , who is …

window size: $w = 1$

	chef
the	2
cat	0
chicken	0
city	0
cook	0

… , the club may also employ a **chef** to prepare and cook food items .

… is up to remy , linguini , and the **chef** colette to cook for many people …

… cooking program the cook and the **chef** with simon bryant , who is …

window size: $w = 4$

	chef
the	3
cat	0
chicken	0
city	0
cook	3

words we are computing vectors for:

context words:	cat	chef	chicken	civic	cooked	council
the	24708	7410	7853	16486	3463	316380
cat	2336	14	23	0	1	36
chicken	23	21	1640	1	181	7
city	116	89	62	943	7	27033
cook	12	113	34	6	34	51

- once we have word vectors, we can compute similarities!
- many ways to define similarity of two vectors
- a simple way: **dot product** (also called inner product):

$$\mathbf{u} \cdot \mathbf{v} = \mathbf{u}^\top \mathbf{v} = \sum_i u_i v_i$$

$\mathbf{u} =$ a vector

$u_i =$ entry i in the vector

- dot product is large when the vectors have very large values in the same dimensions

with dot product as similarity function, let's find the most similar words ("nearest neighbors") to each word:

	cat	chef	chicken	civic	cooked	council
nearest neighbors	council	council	council	council	council	council
	cat	cat	cat	cat	cat	cat
	civic	civic	civic	civic	civic	civic
	chicken	chicken	chicken	chicken	chicken	chicken
	chef	chef	chef	chef	chef	chef
	cooked	cooked	cooked	cooked	cooked	cooked

with dot product as similarity function, let's find the rds ("nearest neighbors") to each word:

	council
the	316380
cat	36
chicken	7
city	27033
cook	51

	chef	chicken	civic	cooked	council
l	council	council	council	council	council
	cat	cat	cat	cat	cat
	civic	civic	civic	civic	civic
n	chicken	chicken	chicken	chicken	chicken
chef	chef	chef	chef	chef	chef
cooked	cooked	cooked	cooked	cooked	cooked

- dot product is large when vectors have large values in same dimensions, doesn't control for vector length
- vector length: $||\mathbf{u}|| = \sqrt{\sum_i u_i^2}$
- **cosine similarity**: $\frac{\mathbf{u}^\top \mathbf{v}}{||\mathbf{u}||\,||\mathbf{v}||}$

$\mathbf{u} =$ a vector

$u_i =$ entry i in the vector

this is the cosine of the angle between the two vectors!

now using cosine similarity:

nearest neighbors

cat	chef	chicken	civic	cooked	council
cat	chef	chicken	civic	cooked	council
chef	civic	cooked	council	chef	civic
cooked	cooked	chef	chef	civic	chef
civic	council	civic	cooked	council	cooked
council	cat	council	cat	cat	cat
chicken	chicken	cat	chicken	chicken	chicken

Improving Counts

- counts of common words (“the”) are very large, but not very useful
- many ways proposed for improving raw counts

Improving Counts

- counts of common words (“the”) are very large, but not very useful
- many ways proposed for improving raw counts
- **tf-idf** (“term frequency-inverse document frequency”)
  - **tf** is just the count of the two words
  - **idf** is computed for the context word
    - document frequency = number of documents in which a word appears
    - inverse of document frequency will be small for common words
  - vector entry is then the product of tf and idf

Pointwise Mutual Information (PMI)

- consider two random variables, X and Y
- do two events $X = x$ and $Y = y$ occur together more often than if they were independent?

$$\mathrm{pmi}(x, y) = \log_2 \frac{p_{X,Y}(x, y)}{p_X(x)\, p_Y(y)}$$

- if they are independent, PMI = 0

PMI for Word Vectors

- for word vectors,
  X is the **center word**
  Y is the **context word**

- each probability can be estimated using counts we already computed!

$\#(x, y)$ = co-occurrence count of x and y

N = total count

$$\text{pmi}(x, y) = \log_2 \frac{p_{X,Y}(x, y)}{p_X(x)\, p_Y(y)}$$

$$p_{X,Y}(x, y) = \frac{\#(x, y)}{N}$$

$$p_X(x) = \frac{\sum_y \#(x, y)}{N}$$

$$p_Y(y) = \frac{\sum_x \#(x, y)}{N}$$

Top co-occurrence counts with “chicken”

14464	,	1525	or	508	pork
7853	the	1225	for	500	meat
6276	and	1061	‘s	481	be
5931	.	940	fried	479	he
5213	a	906	on	452	such
3963	of	889	was	445	his
3282	in	869	that	417	at
2520	to	828	are	405	soup
2438	"	777	by	389	made
2339	is	746	from	384	rice
2127	with	710	it	375	but
1818	(	600	beef	350	has
1745	)	590	which	330	fish
1640	chicken	557	also	325	other
1594	as	531	an	318	this

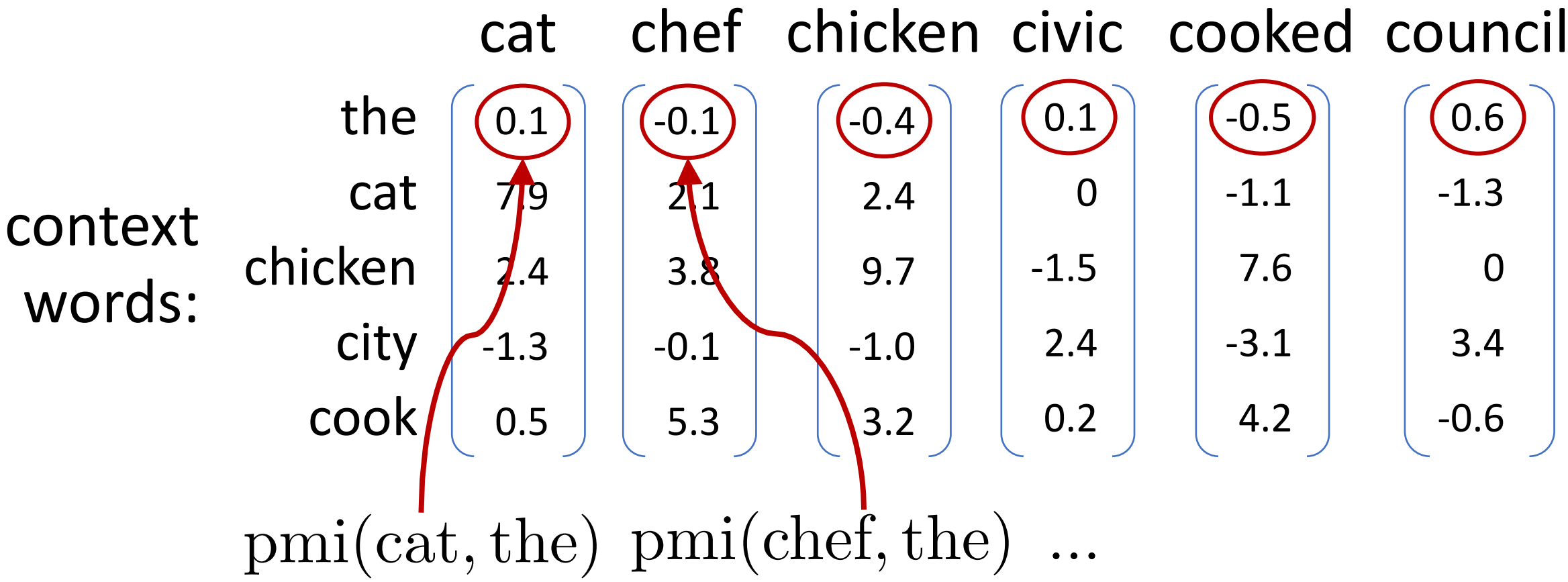
Words with largest PMI with “chicken”

10.2	fried	7.0	robot	6.1	pig
9.7	chicken	6.9	burger	6.0	breeds
9.3	pork	6.8	recipe	6.0	vegetable
9.0	beef	6.6	vegetables	6.0	potato
8.7	soup	6.6	potatoes	5.9	goose
7.8	sauce	6.6	goat	5.9	dixie
7.7	curry	6.5	eggs	5.9	kung
7.6	cooked	6.4	cow	5.9	pie
7.5	lamb	6.4	pizza	5.8	menu
7.4	dish	6.4	rice	5.8	steamed
7.3	shrimp	6.3	ribs	5.8	tastes
7.3	egg	6.3	tomatoes	5.7	beans
7.2	sandwich	6.2	cheese	5.7	butter
7.2	dishes	6.2	duck	5.7	barn
7.2	meat	6.1	chili	5.7	breed

words we are computing vectors for:

context words:	cat	chef	chicken	civic	cooked	council
the	24708	7410	7853	16486	3463	316380
cat	2336	14	23	0	1	36
chicken	23	21	1640	1	181	7
city	116	89	62	943	7	27033
cook	12	113	34	6	34	51

words we are computing vectors for:



using counts:

nearest neighbors

cat	chef	chicken	civic	cooked	council
cat	chef	chicken	civic	cooked	council
chef	civic	cooked	council	chef	civic
cooked	cooked	chef	chef	civic	chef
civic	council	civic	cooked	council	cooked
council	cat	council	cat	cat	cat
chicken	chicken	cat	chicken	chicken	chicken

using PMIs:

nearest neighbors

cat	chef	chicken	civic	cooked	council
cat	chef	chicken	civic	cooked	council
chicken	chicken	cooked	council	chicken	civic
chef	cooked	chef	chef	chef	chicken
cooked	cat	cat	cat	cat	chef
civic	council	council	chicken	council	cooked
council	civic	civic	cooked	civic	cat

Positive PMI?

- some have found benefit by truncating PMI at 0 (“positive PMI”)
- negative PMI: words occur together less than we would expect, i.e., they are anticorrelated
- these anticorrelations may need more data to reliably estimate
- however, negative PMIs do seem reasonable!

Largest PMIs:

10.2	fried
9.7	chicken
9.3	pork
9.0	beef
8.7	soup
7.8	sauce
7.7	curry
7.6	cooked
7.5	lamb
7.4	dish
7.3	shrimp
7.3	egg
7.2	sandwich

PMIs close to zero:

0.003	climbed
0.003	detailing
0.002	turkish
0.002	oaks
0.001	productivity
0.000	swing
-0.001	structures
-0.001	thirteenth
-0.001	commentators
-0.001	palmer
-0.002	obstacles
-0.003	horns
-0.003	burning

Smallest PMIs:

-4.6	users
-4.6	data
-4.7	discussion
-4.7	museum
-4.7	below
-4.8	editors
-4.8	railway
-4.8	committee
-4.8	elected
-4.9	championship
-5.0	archive
-5.3	edits
-6.1	deletion

Summary (1/2)

- **word vectors**: the use of a vector of numbers to represent a word
- **distributional word vectors**: the use of distributional statistics (e.g., word co-occurrence counts) in defining word vectors
- different window sizes lead to different kinds of similarity in the vectors
- many options for computing similarity of two vectors:
  - **dot product** is a simple starting point
  - **cosine similarity** accounts for vector length and works better for word vectors
- simple distributional vectors are ok, but modifying counts can help

Summary (2/2)

- one method: scale down counts for common context words, usually with a variant of **inverse document frequency (idf)**
- another method: **pointwise mutual information (PMI)**

- PMI provides information about whether two events are independent
- view center and context word slots as random variables; words are events/outcomes of those random variables
- use PMI(center word, context word) values in place of counts for better word vectors