EECS 487: Introduction to Natural Language Processing

Instructor: Prof. Lu Wang

Computer Science and Engineering

University of Michigan

https://web.eecs.umich.edu/~wangluxy/

Today’s Outline

- Show me the data!
- What exactly is a word?
- Text normalization

[Modified from slides by Laura Burdick]

Corpora

- Corpus (plural: corpora) = collection of text or speech
- Properties of a corpus:
  - One or more speakers or writers
  - A specific dialect of a specific language
  - At a specific time
  - In a specific place
  - For a specific function
- Pay attention to your corpus! You'll get different results with different corpora

Datasheets for Corpora

- Movement to document some more data about our corpora (also see Gebru et al (2020), Bender and Friedman (2018))
  - Motivation: Why was the corpus collected, by whom, and who funded it?
  - Situation: When and in what situation was the text written/spoken? For example, was there a task? Was the language originally spoken conversation, edited text, social media communication, monologue vs. dialogue?
  - Language variety: What language (including dialect/region) was the corpus in?
  - Speaker demographics: What was, e.g., the age or gender of the text's authors?
  - Collection process: How big is the data? If it is a subsample how was it sampled? Was the data collected with consent? How was the data pre-processed, and what metadata is available?
  - Annotation process: What are the annotations, what are the demographics of the annotators, how were they trained, how was the data annotated?
  - Distribution: Are there copyright or other intellectual property restrictions?

Today’s Outline

- Show me the data!
- What exactly is a word?
- Text normalization

How many words are in this sentence?

- He stepped out into the hall, was delighted to encounter a water fountain.

How many words are in this utterance?

- “I do uh main- mainly business data processing”

Sidebar: *uh* vs. *um*

ELSEVIER

Cognition
Volume 84, Issue 1, May 2002, Pages 73-111

Using *uh* and *um* in spontaneous speaking

Herbert H. Clark [a], Jean E. Fox Tree [b]

"The proposal examined here is that speakers use *uh* and *um* to announce that they are initiating what they expect to be a minor *(uh)*, or major *(um)*, delay in speaking…"

Other quandaries

- Are *They* and *they* different words?

- Are *cat* and *cats* the same word?
  - They have the same lemma *cat*
  - Lemma = a set of lexical forms having the same stem, the same major part-of-speech (POS), and the same word sense
    - E.g., *sang*, *sung*, *sings* -> all have lemma *sing*
  - Wordform = full inflected or derived form of the word

How many words are there in English?

- Two ways to talk about words:
  - Types = number of distinct words in a corpus
  - Tokens = total number of running words
- How many tokens in the following sentence? How many types?
  - They picnicked by the pool, then lay back on the grass and looked at the stars.

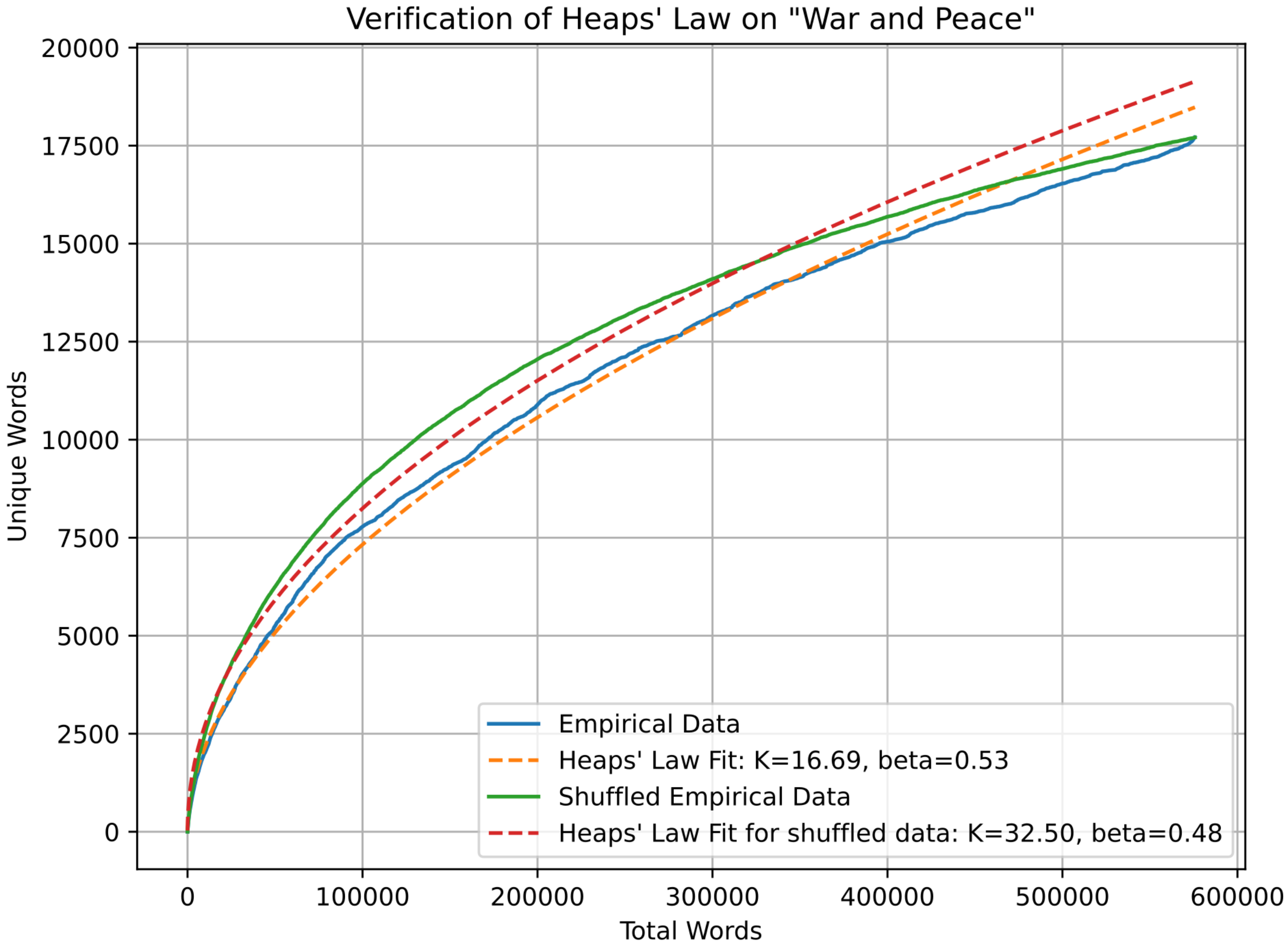
So, how many words are in English?

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

- Relationship between number of types and number of tokens – Herdan's Law or Heaps' Law
  - V = set of words in the vocabulary; $|V|$ = vocabulary size (number of types)
  - N = number of tokens
  - $|V| = kN^{\beta}$, where k and β are positive constants, and $0 < \beta < 1$
  - β depends on the corpus size and genre (above, β goes from .67 to .75)

Herdan’s / Heaps’ Law

Image from https://commons.wikimedia.org/wiki/File:Heaps%27_Law_on_%22War_and_Peace%22.svg#/media/File:Heaps'_Law_on_"War_and_Peace".svg



So, how many words are in English?

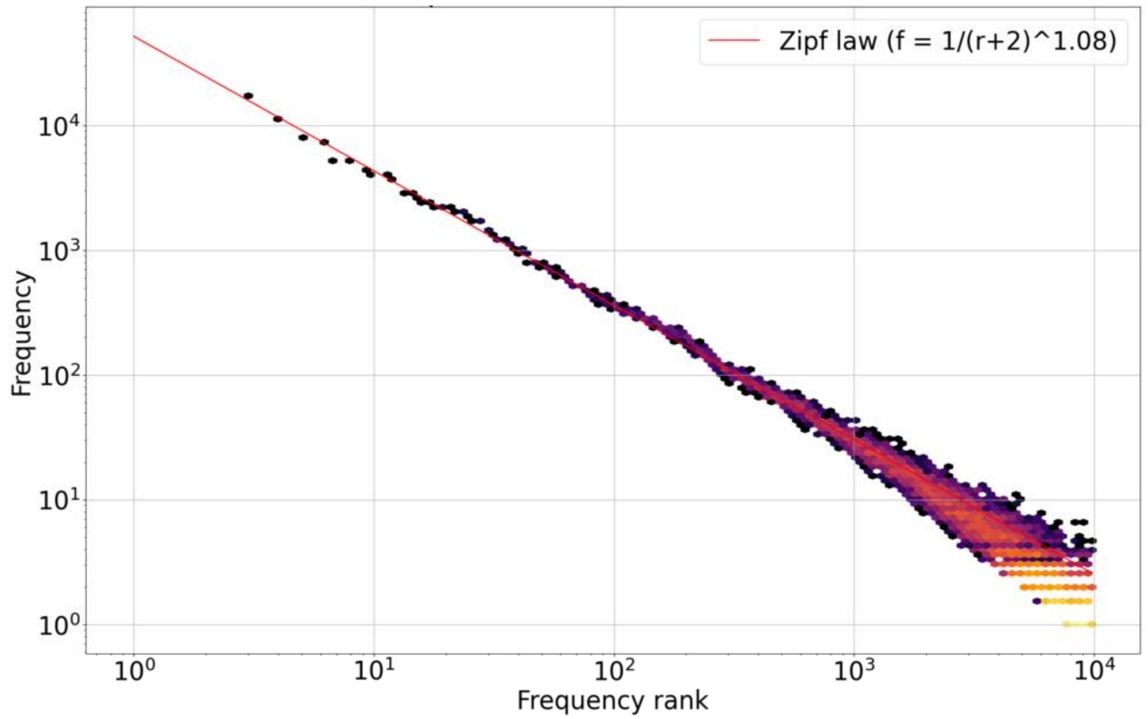
- Instead of counting wordform types, what if we counted lemmas?
  - Dictionary entries or boldface forms give us rough upper bound on number of lemmas
  - 1989 Oxford English Dictionary – how many entries?

Another famous law: Zipf's Law

- Suppose we count up how many times a wordtype appears in a large corpus, and then list the words in order of their frequency of occurrence
- What is the relationship between the frequency of a word f and its position in the list (its rank, r)?
- Zipf's Law: $fr = k$, where k is a constant
  - E.g., the 50th most common word should occur 3 times more frequently than the 150th most common word

Zipf's Law for War and Peace

- Notice:
  - A few very common words
  - A middle number of medium frequency words
  - Many low frequency words



Today’s Outline

- Show me the data!
- What exactly is a word?
- Text normalization

Text Normalization

- We need to “clean up” our text before we can use it
- Common tasks:
  - Tokenizing (segmenting) words
  - Normalizing word formats
  - Segmenting sentences

Word Tokenization

- Word tokenization = the task of segmenting running text into words
- Common standard: Penn Treebank tokenization standard
  - Penn Treebank is a corpora of parsed data released by the Linguistic Data Consortium (LDC)

Input: `"The San Francisco-based restaurant," they said, "doesn't charge $10".`

Output: `"␣The␣San␣Francisco-based␣restaurant␣,␣"␣they␣said␣,␣"␣does␣n't␣charge␣$␣10␣"␣.`

Python Natural Language Toolkit (NLTK)

- NLTK uses a fast algorithm based on 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', '...']
```

What about languages that don’t use spaces?

- Written Chinese, Japanese, Thai, etc. don’t use spaces to mark potential word-boundaries
- Example: Chinese words are composed of characters (hanzi)
  - Each character generally represents a single unit of meaning (a morpheme) and is pronounceable as a single syllabus

What about languages that don't use spaces?

- Consider this sentence: 姚明进入总决赛
  "Yao Ming reaches the finals"
- Could be treated as 3 words: 姚明 进入 总决赛
  YaoMing reaches finals
- Or as 5 words: 姚 明 进入 总 决赛
  Yao Ming reaches overall finals
- Or as 7 words (each character is a "word"): 姚 明 进 入 总 决 赛
  Yao Ming enter enter overall decision game
- Often, for Chinese, we use the last option! But this isn't true in every language

Another option for segmentation

- Instead of
  - White-space segmentation
  - Single-character segmentation
- Use the data to tell us how to tokenize!

- Subword tokenization = tokens can be parts of words as well as whole words

Subword tokenization

- Three common algorithms:
  - Byte-Pair Encoding (BPE) (Sennrich et al., 2016)
  - Unigram language modeling tokenization (Kudo, 2018)
  - WordPiece (Schuster and Nakajima, 2012)
  - Also a SentencePiece library that includes implementations of the first 2 algorithms (Kudo and Richardson, 2018)
- All have 2 parts:
  - A token learner - takes a raw training corpus and induces a vocabulary (a set of tokens)
  - A token segmenter - takes a raw test sentence and tokenizes it according to that vocabulary

Byte Pair Encoding (BPE) token learner

- Let vocabulary be the set of all individual characters

  = {A, B, C, D,…, a, b, c, d….}

- Repeat:
  - Choose the two symbols that are most frequently adjacent in the training corpus (say 'A', 'B')
  - Add a new merged symbol 'AB' to the vocabulary
  - Replace every adjacent 'A' 'B' in the corpus with 'AB'.
- Until *k* merges have been done.

Byte Pair Encoding (BPE) token learner

function BYTE-PAIR ENCODING(strings C, number of merges k) **returns** vocab V

$V \leftarrow$ all unique characters in C # initial set of tokens is characters
for $i = 1$ **to** k **do** # merge tokens til k times
  $t_L, t_R \leftarrow$ Most frequent pair of adjacent tokens in C
  $t_{NEW} \leftarrow t_L + t_R$ # make new token by concatenating
  $V \leftarrow V + t_{NEW}$ # update the vocabulary
  Replace each occurrence of t_L, t_R in C with t_{NEW} # and update the corpus
return V

Byte Pair Encoding (BPE) token learner

- Usually run inside words (not merging across word boundaries)
- So, first we need to white-space separate the input corpus
  - We add a special end-of-word symbol '__' before space in training corpus

BPE token learner

Original (very fascinating) corpus:

low low low low low lowest lowest newer newer newer newer newer newer wider wider wider new new

Add end-of-word tokens, resulting in this vocabulary:

vocabulary

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

BPE token learner

corpus		**vocabulary**
5	l o w _	_, d, e, i, l, n, o, r, s, t, w
2	l o w e s t _	
6	n e w e r _	
3	w i d e r _	
2	n e w _	

Merge e r to er

corpus		**vocabulary**
5	l o w _	_, d, e, i, l, n, o, r, s, t, w, er
2	l o w e s t _	
6	n e w er _	
3	w i d er _	
2	n e w _	

What's the next merge we make?

BPE

corpus		**vocabulary**
5	l o w _	_, d, e, i, l, n, o, r, s, t, w, er
2	l o w e s t _	
6	n e w er _	
3	w i d er _	
2	n e w _	

Merge er _ to er_

corpus		**vocabulary**
5	l o w _	_, d, e, i, l, n, o, r, s, t, w, er, er_
2	l o w e s t _	
6	n e w er_	
3	w i d er_	
2	n e w _	

What's the next merge we make?

BPE

corpus		vocabulary
5	l o w _	_, d, e, i, l, n, o, r, s, t, w, er, er_
2	l o w e s t _	
6	n e w er_	
3	w i d er_	
2	n e w _	

Merge n e to ne

corpus		vocabulary
5	l o w _	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne
2	l o w e s t _	
6	ne w er_	
3	w i d er_	
2	ne w _	

BPE

The next merges are:

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_

BPE token segmenter algorithm

- Once we've learned our vocabulary, the token segmenter tokenizes a test sentence
- On the test data, run each merge learned from the training data:
  - Greedily
  - In the order we learned them
  - (test frequencies don't play a role, just training frequencies)
- So: merge every e r to er, then merge er _ to er_, etc.
- Result:
  - Test set "n e w e r _" would be tokenized as a full word
  - Test set "l o w e r _" would be two tokens: "low er_"
  - Exercise: What would "w i n n e r _" be tokenized as?

BPE

- In real settings, many thousands of merges on very large corpus
- Most words will be represented as full symbols
- Only very rare words (and unknown words) will have to be represented by their parts

Word Normalization

- Word normalization = 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 = reduce all letters to lower case
  - When is this helpful? When is it not?

Lemmatization

- Lemmatization = represent all words as their lemma, their shared root
  - *am, are, is* $\rightarrow$ *be*
  - *car, cars, car's, cars'* $\rightarrow$ *car*
  - Spanish quiero ('I want'), quieres ('you want') $\rightarrow$ querer 'want'
- *He is reading detective stories* $\rightarrow$ *He be read detective story*

- Most sophisticated lemmatization done using morphological parsing
  - *cats* $\rightarrow$ *cat* and *s*
  - Spanish *amaren* ('if in the future they would love') $\rightarrow$ morpheme *amar* 'to love', and the morphological features are *3PL (*third person plural*)* and *future subjunctive*

Stemming

- Lemmatization algorithms can be complex!
- Simpler but cruder method: stemming
  - Basically chop off all word-final affixes

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

.

Porter Stemmer

- Based on a series of rewrite rules run in series
  - Output of each pass fed as input to the next pass
- Some sample rules:

$$
\begin{aligned}
\text{ATIONAL} &\rightarrow \text{ATE} \quad \text{(e.g., relational} \rightarrow \text{relate)} \\
\text{ING} &\rightarrow \epsilon \quad \text{if stem contains vowel (e.g., motoring} \rightarrow \text{motor)} \\
\text{SSES} &\rightarrow \text{SS} \quad \text{(e.g., grasses} \rightarrow \text{grass)}
\end{aligned}
$$

Stemming

- Simple stemmers tend to commit errors of both over- and under-generalizing (Krovetz, 1993):

Errors of Commission		**Errors of Omission**	
organization	organ	European	Europe
doing	doe	analyzes	analysis
numerical	numerous	noisy	noise
policy	police	sparsity	sparse

Sentence Segmentation

- !, ? mostly unambiguous but period “.” is very ambiguous
  - Sentence boundary
  - Abbreviations like Inc. or Dr.
  - Numbers like .02% or 4.3
- Common algorithm:
  - First decide (using rules of machine learning) whether a period is part of the word or is a sentence-boundary marker (an abbreviation dictionary can help)
  - Then, use a set of rules to split sentences