WHENEVER I LEARN A NEW SKILL I CONCOCT ELABORATE FANTASY SCENARIOS WHERE IT LETS ME SAVE THE DAY.
OH NO! THE KILLER MUST HAVE FOLLOWED HER ON VACATION!
BUT TO FIND THEM WE'D HAVE TO SEARCH THROUGH 200 MB OF EMAILS LOOKING FOR SOMETHING FORMATTED LIKE AN ADDRESS!
IT'S HOPELESS!
EVERYBODY STAND BACK.
I KNOW REGULAR EXPRESSIONS.
PERL!
TAP TAP

https://xkcd.com/208/

CS 2731 / ISSP 2230
Introduction to Natural Language Processing

Session 2: Text normalization

Michael Miller Yoder

January 10, 2023

University of Pittsburgh | School of Computing and Information

Overview: Text normalization

- Course logistics
- Basic terminology
- Regular expressions
- Text normalization

Course logistics

- Reading for today was Jurafsky & Martin sections 2-2.4, 2.6
- **First reading quiz is due next Wed, Jan 17 at 1pm before class**
- Project survey due next Thursday, Jan 18 at 11:59pm
    - See project description
- Project groups will often be 3-4 students instead of 2
- Please remind me of your name before asking or answering a question (just this class session)

NLP terminology: words and corpora

How many words in this phrase?

they lay back on the San Francisco grass and looked at the stars and their

- How many?
  - 15 tokens (or 14 if you count "San Francisco" as one)
  - 13 types (or 12) (or 11?)
- **Type**: a unique word in the vocabulary
- **Token**: an instance of a word type in running text
- **Lemma**: same stem, part of speech, rough word sense
  - cat and cats = same lemma
- **Wordform**: the full inflected surface form
  - cat and cats = different wordforms

Slide adapted from Jurafsky & Martin

How many words in a corpus?

Corpus: a (machine-readable) collection of texts

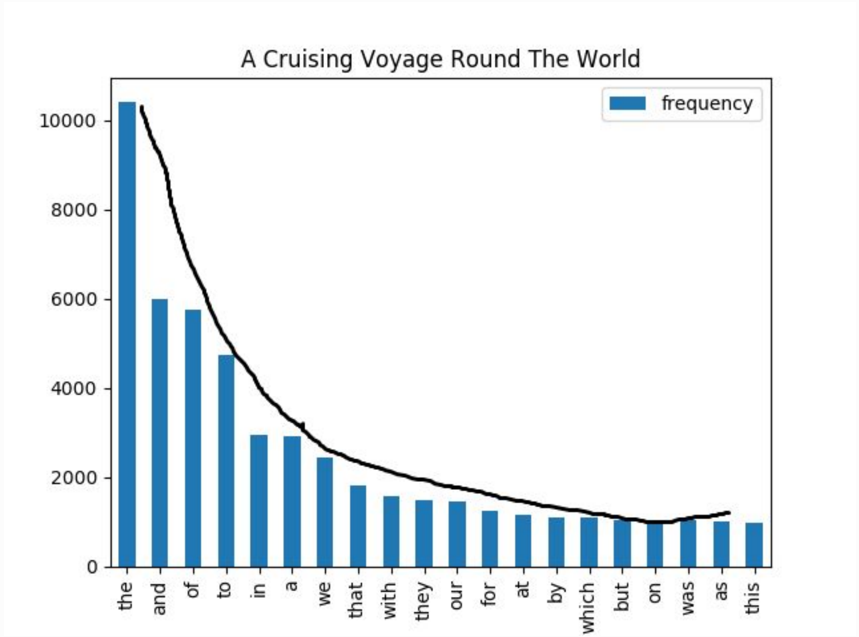
N = number of tokens

V = vocabulary = set of types, $|V|$ is size of vocabulary

	Tokens = N	Types = |V|
Switchboard phone conversations	2.4 million	20 thousand
Shakespeare	884,000	31 thousand
COCA	440 million	2 million
Google N-grams	1 trillion	13+ million

Slide adapted from Jurafsky & Martin

Word frequencies: Zipf's Law



The Lexical Learner blog

- Word (type) frequency is inversely proportional to word frequency rank

$$\text{frequency} \propto \frac{1}{(\text{rank} + b)^a}$$

- "Long tail" of infrequent words

Corpora vary along dimensions like

- Texts don't appear out of nowhere!
- **Language**: 7097 languages in the world
- **Variety**, like African American Language varieties.
  - AAE Twitter posts might include forms like "*iont*" *(I don't)*
- **Code switching**, e.g., Spanish/English, Hindi/English:

  Por primera vez veo a @username actually being helpful! It was beautiful:)

  [For the first time I get to see @username actually being helpful! it was beautiful:)]

  dost tha or ra- hega ... dont wory ... but dherya rakhe

  [“he was and will remain a friend ... don’t worry ... but have faith”]

- **Genre:** newswire, fiction, scientific articles, Wikipedia
- **Author Demographics**: writer's age, gender, ethnicity, SES
- Corpus datasheets [Bender & Friedman 2018, Gebru+ 2020] ask about this information

Slide adapted from Jurafsky & Martin

Regular expressions (regex)

Regular expressions

- A formal language for specifying text strings
- How can we search for any of these?
    - woodchuck
    - woodchucks
    - Woodchuck
    - Woodchucks

Slide adapted from Jurafsky & Martin

Regular Expressions: Disjunctions (OR)

- Letters inside square brackets []

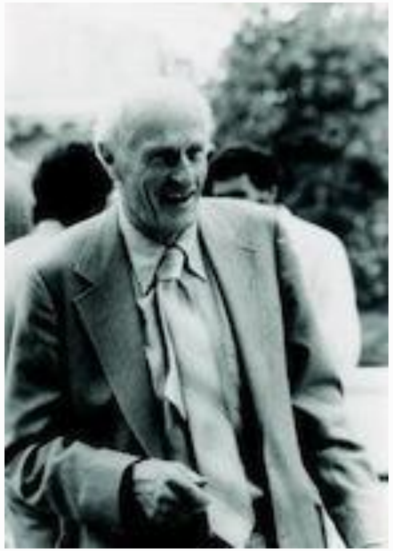
Pattern	Matches
`[wW]oodchuck`	Woodchuck, woodchuck
`[1234567890]`	Any digit

- Ranges `[A-Z]` `[a-z]` `[0-9]`
- Negations `[^A-Z]`
    - Carat means negation only when first in []
- Sequence disjunctions with pipe |
    - `groundhog|woodchuck`

Slide adapted from Jurafsky & Martin

Regular Expressions wildcards: *+.

Pattern	Matches	
`oo*h`	0 or more of previous char	oh ooh oooh ooooh
`o+h`	1 or more of previous char	oh ooh oooh ooooh
`beg.n`	Any char	begin begun begun beg3n


Stephen C Kleene

Slide adapted from Jurafsky & Martin

Finite state automata (briefly)

A sheep language

$$\begin{tikzcd}
\arrow[r] & q_0 \arrow[r, bend left, "b"] & q_1 \arrow[r, bend left, "a"] & q_2 \arrow[r, bend left, "a"] & q_3 \arrow[loop above, "a"] \arrow[r, bend left, "!"] & q_4
\end{tikzcd}$$

Recognizes:

- baa!
- baaa!
- baaaa!

Rejects:

- ba
- ba!
- baaa

- When you follow such a transition, the symbol is "consumed"
- If consuming all of the symbols coincides with being at an accepting state, you win! (The FSA accepts the string).
- Otherwise, you lose! (The FSA rejects the string).

Slide credit: David Mortensen

Regular expression example

- Find all instances of the word “the” in a text.

```
the
```

- Misses capitalized examples

```
[tT]he
```

- Incorrectly returns "other" or "theology"

```
[^a-zA-Z][tT]he[^a-zA-Z]
```

Slide adapted from Jurafsky & Martin

Errors

The process we just went through was based on fixing two kinds of errors:

1. Matching strings that we should not have matched (there, then, other)

   False positives (Type I errors)

2. Not matching things that we should have matched (The)

   False negatives (Type II errors)

Capture groups and regular expression substitution

- Say we want to put angles around all numbers after the word *the*:

  the 35 boxes □ the <35> boxes

- Use parens () to "capture" a pattern group and save to a numbered register \1

```
the ([0-9]+)
```

- Can substitute something for the group

  In Python:

```
re.sub(r'the ([0-9]+)', 'the <\1>', input_text)
```

Slide adapted from Jurafsky & Martin

Simple Application: ELIZA

- Early NLP system that imitated a Rogerian psychotherapist [Weizenbaum 1966]
- Uses pattern matching to match phrases

  `“I need X”`

- and translates them into, e.g.

  `“What would it mean to you if you got X?`

Simple Application: ELIZA

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

Slide adapted from Jurafsky & Martin

How ELIZA works

```
.* I’M (depressed|sad) .* → I AM SORRY TO HEAR YOU ARE \1

.* all .* → IN WHAT WAY?

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

Slide adapted from Jurafsky & Martin

Regular expressions summary

- Regular expressions play a surprisingly large role in NLP
    - Sophisticated sequences of regular expressions are often the first model for any text processing text
- For hard tasks, we use machine learning classifiers
    - But regular expressions are still used for pre-processing, or used to extract features for the classifiers

Slide adapted from Jurafsky & Martin

Text normalization (preprocessing)

Every NLP task requires text normalization

1. Tokenizing (separating) words
2. Normalizing word formats
3. Segmenting sentences

Slide adapted from Jurafsky & Martin

Tokenization

Space-based tokenization

- A very simple way to tokenize
- For languages that use space characters between words
    - Arabic, Cyrillic, Greek, Latin, etc., based writing systems
- Segment off a token between instances of spaces

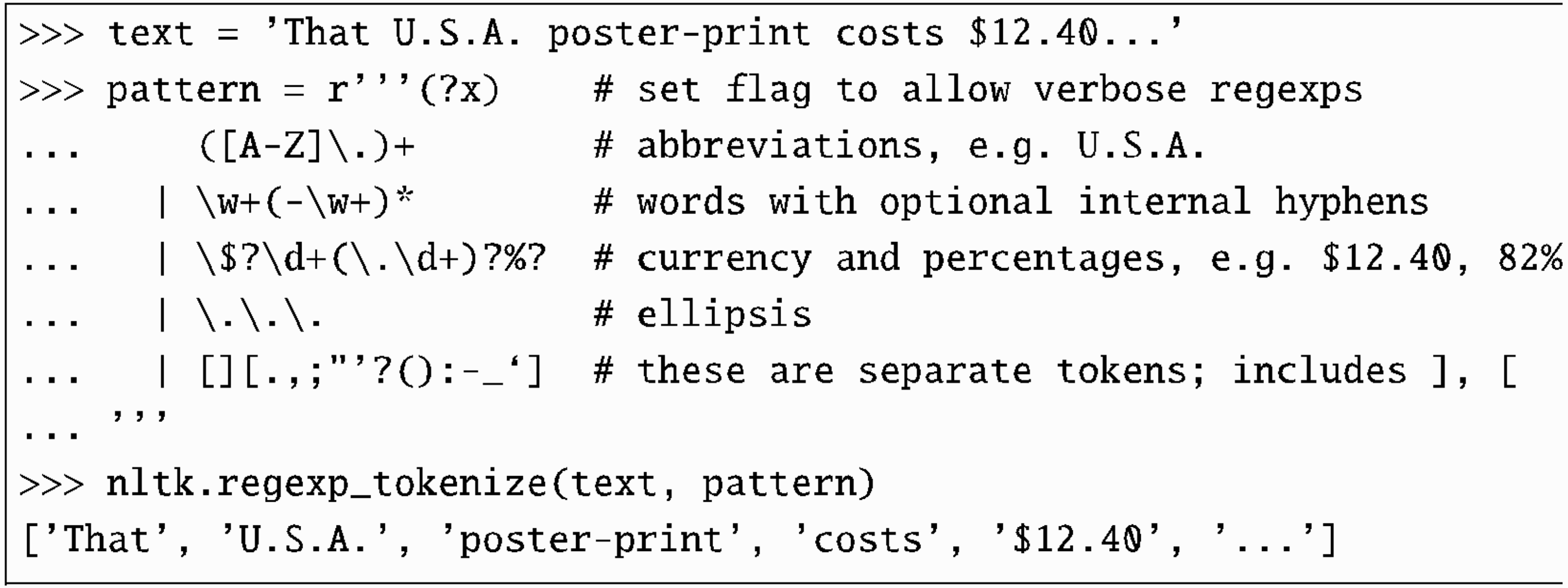
Slide adapted from Jurafsky & Martin

Issues in Tokenization

- 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.pitt.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

Slide adapted from Jurafsky & Martin

Regex-based tokenization



```
>>> 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 and 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’, ’...’]
```

- NLTK [Bird+ 2009] provides regex and ML models for tokenization (like punkt tokenizer)
- spaCy, other packages provide good tokenization

Slide adapted from Jurafsky & Martin

Tokenization in languages without spaces between words

- Many languages (like Chinese, Japanese, Thai) don't use spaces to separate words!
- How do we decide where the token boundaries should be?

Slide adapted from Jurafsky & Martin

Word tokenization in Chinese

- Chinese words are composed of characters called "**hanzi**" (or sometimes just "**zi**")
- Each one represents a meaning unit called a morpheme
- Each word has on average 2.4 of them.
- But deciding what counts as a word is complex and not agreed upon.

Slide adapted from Jurafsky & Martin

How to do word tokenization in Chinese?

姚明进入总决赛 “Yao Ming reaches the finals”

3 words?

姚明	进入	总决赛
YaoMing	reaches	finals

5 words?

姚	明	进入	总	决赛
Yao	Ming	reaches	overall	finals

7 characters? (don't use words at all):

姚 明 进 入 总 决 赛

Yao Ming enter enter overall decision game

Word tokenization / segmentation

- In Chinese NLP it's common to just treat each character (zi) as a token.
  - So the **segmentation** step is very simple
- In other languages (like Thai and Japanese), more complex word segmentation is required.
  - The standard algorithms are neural sequence models trained by supervised machine learning.

Slide adapted from Jurafsky & Martin

Subword tokenization & BPE

Another option for text tokenization

- **Use the data** to tell us how to tokenize.
- **Subword tokenization** (because tokens can be parts of words as well as whole words)
- Many modern neural NLP systems (like BERT) use this to handle unknown words
- 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

Slide adapted from Jurafsky & Martin

Byte Pair Encoding [BPE, Sennrich+ 2016] 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.

Slide adapted from Jurafsky & Martin

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

Split on whitespace, add end-of-word tokens _

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

Slide adapted from Jurafsky & Martin

BPE token learner

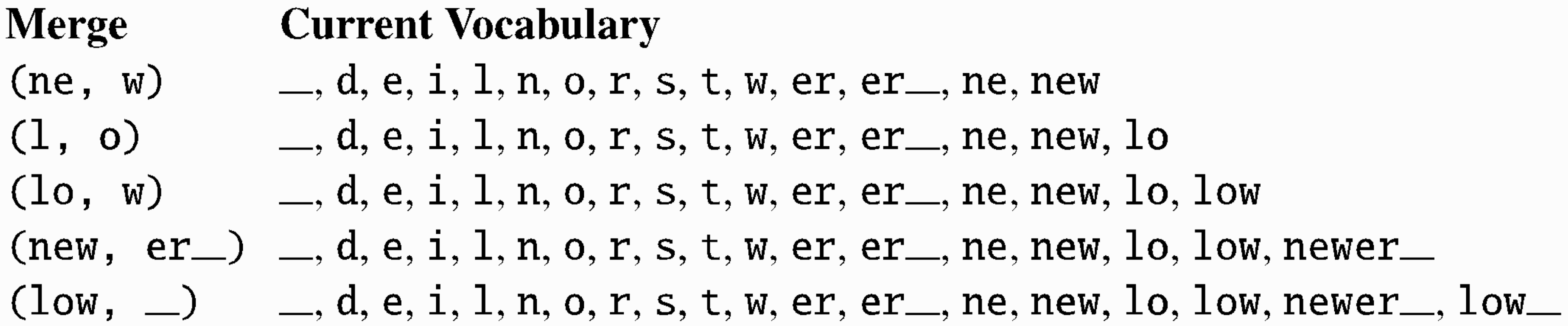
- Merge e r to er

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 _	

- Merge er _ to er_
- Merge n e to ne

BPE token learner

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

- On the test data, run each merge learned from the training data:
  - Greedily, in the order we learned them
- 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_"

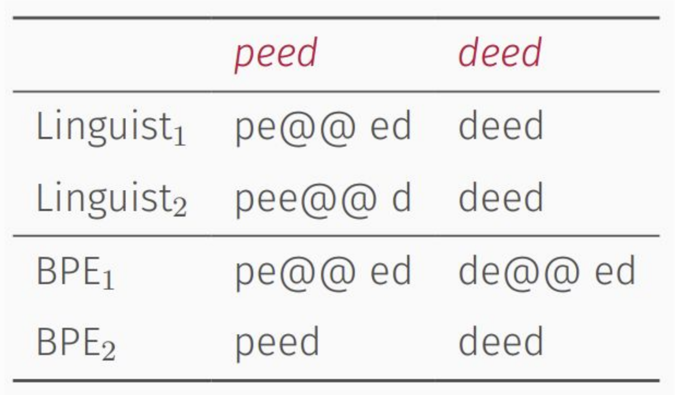
Properties of BPE tokens

Usually include:

- frequent words
- frequent subwords

Which are often morphemes (meaningful word units) like -*est* or –*er*

- But are often not, too! (@@ is a token break)



	peed	*deed*
Linguist$_1$	pe@@ ed	deed
Linguist$_2$	pee@@ d	deed
BPE$_1$	pe@@ ed	de@@ ed
BPE$_2$	peed	deed

Slide adapted from Jurafsky & Martin, David Mortensen

Other preprocessing

Case folding (lowercasing)

- Applications like IR: reduce all letters to lowercase
    - Since users tend to use lowercase
    - 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)

Slide adapted from Jurafsky & Martin

Lemmatization

Represent 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*

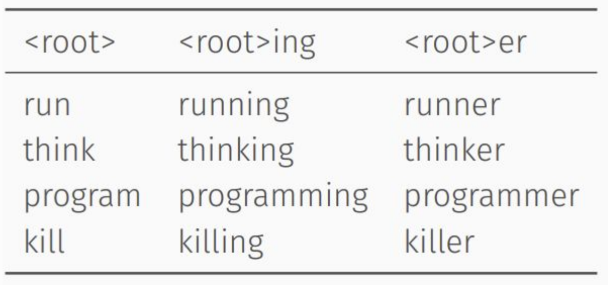
Slide adapted from Jurafsky & Martin

Lemmatization is done by Morphological Parsing

- Morphemes: small meaningful units that make up words
  - Roots: The core meaning-bearing units
  - Affixes: Parts that adhere to roots

un-think-able; kitten-s

- Affixes can add grammatical meaning (inflections, 2nd column) or modify semantic meaning (derivations, 3rd column)



<root>	<root>ing	<root>er
run	running	runner
think	thinking	thinker
program	programming	programmer
kill	killing	killer

Slide adapted from Jurafsky & Martin, David Mortensen

Lemmatization is done by Morphological Parsing

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

Dealing with complex morphology is necessary for many languages

- e.g., the Turkish word:
  Uygarlastiramadiklarimizdanmissinizcasina
  '(behaving) as if you are among those whom we could not civilize'

  Uygar 'civilized' + las 'become'
  + tir 'cause' + ama 'not able'
  + dik 'past' + lar 'plural'
  + imiz '1pl' + dan 'abl'
  + mis 'past' + siniz '2pl' + casina 'as if'

Slide adapted from Jurafsky & Martin

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

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

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

Slide adapted from Jurafsky & Martin

Stopword removal

- Do we want to keep "function words" like *the*, *of*, *and*, *I*, *you*, etc?
- Sometimes no (information retrieval)
- Sometimes yes (authorship attribution)

Sentence segmentation

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

- Sentence boundary
- Abbreviations like Inc. or Dr.
- Numbers like .02% or 4.3

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 (period as a single token is an indication of a sentence boundary, e.g.).

Slide adapted from Jurafsky & Martin

Conclusion and example scenarios

Conclusion: Text normalization

- Regular expressions match flexible sequences of characters and allow substitution of groups of characters
- Tokenization: splitting texts into sequences of words
  - Subword tokenization finds tokens based on frequencies of sequences of characters in data
- Lemmatization: normalizing words to their dictionary roots
- Stemming: chopping off affixes of words to reduce them to stems
- Stopwords are function words like “the”, “a”, “and”, “of”, etc that are often ignored in NLP applications

Preprocessing decisions: example scenarios

- Build a Chinese - French machine translation system
- Study what topics are generally discussed on an online forum through what words people commonly use
- Extract prices from a stock ticker
- Build a dialogue agent in Turkish

Preprocessing considerations:

- Tokenization issues?
- Lowercasing/case folding?
- Stem/lemmatize?
- Morphological analysis needed?
- Use regular expressions?

Questions?

Enjoy MLK Day holiday

No class on Monday
First reading quiz due next Wed Jan 17 at 1pm
Project survey due next Thu Jan 18 at 11:59pm

Edit distance

How similar are two text strings?

Spell correction

- The user typed “graffe”
  - Which is closest?
- graf
- graft
- grail
- giraffe

Computational Biology

- Align two sequences of nucleotides

```
AGGCTATCACCTGACCTCCAGGCCGATGCCC
TAGCTATCACGACCGCGGTCGATTTGCCCGAC
```

- Resulting alignment:

```
-AGGCTATCACCTGACCTCCAGGCCGA--TGCCC---
TAG-CTATCAC--GACCGC--GGTCGATTTGCCCGAC
```

Also for Machine Translation, Information Extraction, Speech Recognition

Slide adapted from Jurafsky & Martin

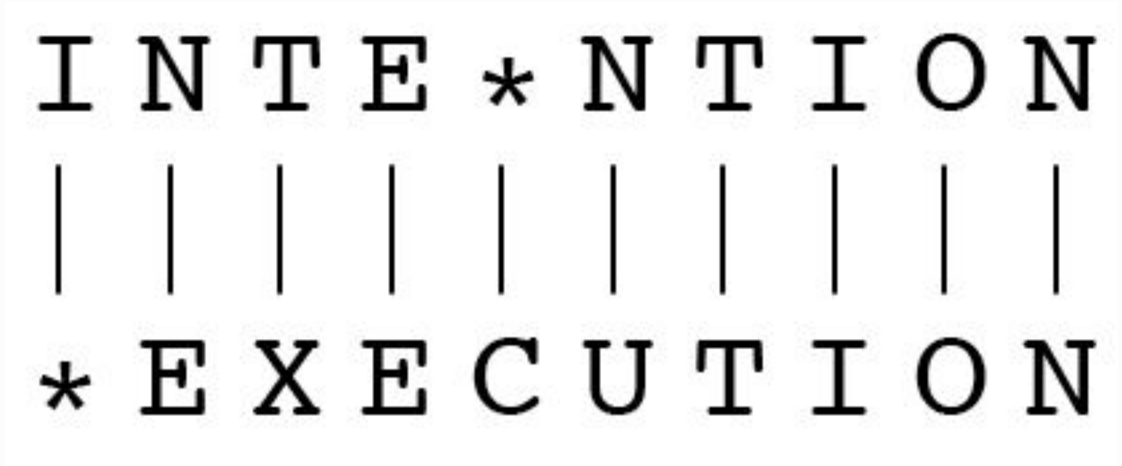
Edit distance

- The minimum edit distance between two strings
- Is the minimum number of editing operations
    - Insertion
    - Deletion
    - Substitution
- Needed to transform one into the other

Slide adapted from Jurafsky & Martin

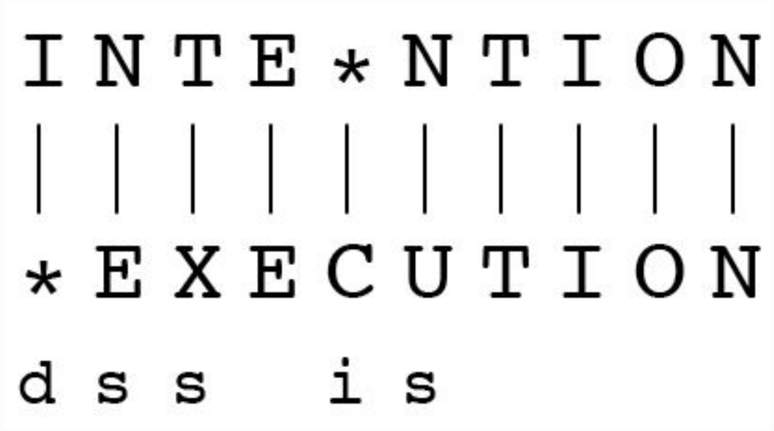
Minimum edit distance

- Two strings and their **alignment**:



Minimum edit distance

- If each operation has cost of 1
  - Distance between these is 5
- If substitutions cost 2 (Levenshtein)
  - Distance between them is 8



Slide adapted from Jurafsky & Martin

How to find the minimum edit 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

Minimum edit as search

- But the space of all edit sequences is huge!
  - 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 intermediate states.

Slide adapted from Jurafsky & Martin

Dynamic Programming for Minimum Edit Distance

- **Dynamic programming**: A tabular computation of D(*n*,*m*)
    - Solving problems by combining solutions to subproblems.
    - Bottom-up
- For two strings: X of length *n*, Y of length *m*
- We define D(*i*,*j*)
    - the edit distance between X[1..*i*] and Y[1..*j*]
        - i.e., the first *i* characters of X and the first *j* characters of Y
    - The edit distance between X and Y is thus **D(*n*,*m*)**
- 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* ($0 < j < m$)

Slide adapted from Jurafsky & Martin

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

Slide adapted from Jurafsky & Martin

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

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

Slide adapted from Jurafsky & Martin

The edit distance table

$D(n, m) = 8$

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

Slide adapted from Jurafsky & Martin

Performance

Time:

$O(nm)$

Space:

$O(nm)$

Slide adapted from Jurafsky & Martin