

Natural Language Processing: Basic Text Processing

14 February 2022

Agenda

1. **Regular expressions:** Important tool for describing text patterns
2. **Text normalization:** Converting text to convenient, standard form
 - Tokenization + Zipf/Heaps Laws
 - Lemmatization, stemming, stopping
 - Sentence segmentation
 - HTML cleaning
3. **Byte Pair Encoding & Edit distance**

Basic Text Processing

Regular Expressions

Regular expressions

- A formal language for specifying text strings
- How can one search for any of these?
 - elephant
 - elephants
 - Elephant
 - Elephants



Regular expressions

- Practical language for specifying text strings, used in every computer language, word processor and text processing tools (e.g., grep or Emacs)
- Especially useful for searching, given particular patterns and a corpus (a document or document collection)
 - **Corpus:** computer-readable text or speech
- Simplest case: sequence of simple characters:
 - Expression `/able/` matches any string containing substring “able”
 - It can be also an individual character `/!/`

Regular Expressions: Disjunctions

- Square brackets `[]` specify disjunction

Pattern	Matches
<code>[wW]oodchuck</code>	Woodchuck, woodchuck
<code>[1234567890]</code>	Any digit

- Ranges `[A-Z]`

Pattern	Matches	
<code>[A-Z]</code>	An upper case letter	<u>D</u> renched Blossoms
<code>[a-z]</code>	A lower case letter	<u>m</u> y beans were tasty
<code>[0-9]</code>	A single digit	Chapter <u>1</u> : Down the Rabbit Hole
<code>[b-f]</code>	Any letter: b, c, d, e, f	Dr <u>e</u> nched Blossoms

Regular Expressions: Negation in Disjunction

- Negations `[^Ss]`
 - Carat means negation only when is mentioned first in []

Pattern	Matches	
<code>[^A-Z]</code>	Not an upper case letter	O <u>y</u> fn pripetchik
<code>[^Ss]</code>	Neither 'S' nor 's'	<u>I</u> have no exquisite reason"
<code>[^.]</code>	Not a dot	<u>D</u> r. Lee is there.
<code>[^e^]</code>	Neither e nor ^	e <u>v</u> erything is great
<code>a^b</code>	The pattern "a carat b"	Look up <u>a^b</u> now

Regular Expressions: More Disjunction

- The pipe `|` for disjunction is needed to specify disjunctions of strings

Pattern	Matches
<code>groundhog woodchuck</code>	
<code>yours mine</code>	yours mine
<code>a b c</code>	= <code>[abc]</code>
<code>[gG]roundhog [Ww]oodchuck</code>	

Regular Expressions: ? * + .

Pattern	Matches	
<code>colou?r</code>	Optional (0 or 1) previous char	<u>color</u> <u>colour</u>
<code>oo*h!</code>	0 or more of previous char	<u>oh!</u> <u>ooh!</u> <u>oooh!</u> <u>ooooh!</u>
<code>o+h!</code>	1 or more of previous char	<u>oh!</u> <u>ooh!</u> <u>oooh!</u> <u>ooooh!</u>
<code>baa+</code>		<u>baa</u> <u>baaa</u> <u>baaaa</u> <u>baaaaa</u>
<code>beg.n</code>	Any character	<u>begin</u> <u>begun</u> <u>begun</u> <u>beg3n</u>
<code>/aardvark.* aardvark/</code>	any line in which aardvark appears twice.	<u>aardvark is a complex word.</u> <u>aardvark</u>



Stephen C Kleene

Kleene *, Kleene +

Regular Expressions: More Disjunction

- Sometimes we need to group characters in parenthesis to make it act as a single character or to scope |:
 - `/gupp(y|ies)/` for disjunction of only suffixes “y” and “ies”
 - Suppose we want to match repeated string: Column 1 Column 2 Column 3
The expression `/Column [0-9]+ */` will not work but `/[Column [0-9]+]*/` will do

Regular Expressions: Anchors [^] ^{\$}

- Special characters to anchor regular expressions to particular places in a string

Pattern	Matches
[^] [A-Z]	<u>P</u> alo Alto
[^] [^A-Za-z]	<u>1</u> "Hello"
\. ^{\$}	The end <u>.</u>
[.] ^{\$}	The end <u>?</u> The end <u>!</u>
\ ^b the\ ^b	<u>the</u> car other

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]`

`[^a-zA-Z]` implies that there must be some
single (although non-alphabetic) character

`(^[^a-zA-Z][tT]he[^a-zA-Z]|$)`

Errors

- The refinement process in the previous slide was based on fixing two kinds of errors
 - Matching strings that should not be matched (there, then, other)
 - False positives (Type I)
 - Not matching things that we should have matched (The)
 - False negatives (Type II)

Errors cont.

- In NLP we are always dealing with these kinds of errors
- Reducing the error rate for an application often involves two antagonistic efforts:
 - Increasing precision (minimizing false positives)
 - Increasing coverage or recall (minimizing false negatives)

Substitutions

- Substitution in Python and UNIX commands:
- `s/regex1/pattern/`
- e.g.:
- `s/colour/color/`

Capture Groups

- Say we want to put angles around all numbers:

the 35 boxes → *the <35> boxes*

- Use parentheses () to "capture" a pattern into a numbered register (1, 2, 3...)
- Use \1 to refer to the contents of the register

s / ([0-9] +) / <\1> /

Capture groups: multiple registers

- `/the (.*)er they (.*) , the \1er we \2/`

- Matches

the faster they ran, the faster we ran

- But not

the faster they ran, the faster we ate

But suppose we don't want to capture?

- Parentheses have a double function: grouping terms, and capturing
- Non-capturing groups: add a `?:` after paren:
- `/ (?:some|a few) (people|cats) like some \1/`
- `matches`
 - `some cats like some cats`
- `but not`
 - `some cats like some a few`

Simple Application: ELIZA

- Early NLP system that imitated a Rogerian psychotherapist (Weizenbaum, 1966)
- Uses pattern matching to match, e.g.,:
 - "I need X"and translates them into, e.g.
 - "What would it mean to you if you got X?"

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

How ELIZA works?

- s/. * I'M (depressed|sad) . */I AM SORRY TO HEAR YOU ARE \1/
- s/. * I AM (depressed|sad) . */WHY DO YOU THINK YOU ARE \1/
- s/. * all . */IN WHAT WAY?/
- s/. * always . */CAN YOU THINK OF A SPECIFIC EXAMPLE?/

Example of Eliza in action

User: I am unhappy.

ELIZA: DO YOU THINK COMING HERE WILL HELP YOU NOT TO BE UNHAPPY

User: I need some help, that much seems certain.

ELIZA: WHAT WOULD IT MEAN TO YOU IF YOU GOT SOME HELP

User: Perhaps I could learn to get along with my mother.

ELIZA: TELL ME MORE ABOUT YOUR FAMILY

User: My mother takes care of me.

ELIZA: WHO ELSE IN YOUR FAMILY TAKES CARE OF YOU

User: My father.

ELIZA: YOUR FATHER

User: You are like my father in some ways.

Summary

- Regular expressions play a surprisingly large role
 - Sophisticated sequences of regular expressions are often the first model for any text processing text
- For many hard tasks, we use machine learning classifiers
 - But regular expressions can be used for preprocessing or as features in the classifiers
 - Can be very useful in capturing generalizations

Basic Text Processing

Word tokenization

Text Normalization

- Nearly every NLP task needs to do text normalization:
 1. Segmenting/tokenizing words in running text
 2. Normalizing word formats
 3. Segmenting sentences in running text

How many words?

- “The University of Innsbruck is a university located in the capital of Tyrol.”
 - 13
- “I do uh main- mainly business data processing”
 - Fragments, filled pauses (fillers) – can be considered as words in some cases (e.g., for speech recognition systems, speaker identification)
- “Seuss’s **cat** in the hat is different from other **cats**!”
 - **Lemma**: canonical, dictionary (or citation) form of a word
 - **cat** and **cats** have same lemma
 - **Wordform**: the full inflected surface form
 - **cat** and **cats** are different wordforms

How many words?

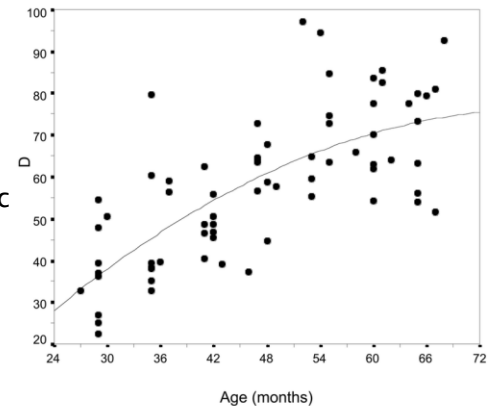
“they lay back on the San Francisco grass and look at the stars and their”

- **Type:** an element of the vocabulary
- **Token:** an instance of that type in running text
(A standard word count tells the number of tokens in text)
- How many tokens and types in the sentence at the top?
 - 15 tokens (or 14)
 - 13 types (or 12)

Type-Token Ratio of Text

- $V(d)/N(d)$
- An index of lexical diversity (different from syntactic complexity), often used to measure text complexity or vocabulary richness
- Can be used for instance for analysis of freshman compositions, studies of childhood acquisition of language, etc.

(Example) Scatter plot of age and lexical diversity (D), with quadratic regression line, for typically developing Cantonese-speaking children (N = 70) [Klee 2004]



How large is vocabulary of English (or any other language)?

N = number of tokens

V = vocabulary = set of types

$|V|$ is the size of the vocabulary

Church and Gale (1990): $|V| > O(N^{1/2})$

Corpora	Tokens = N	Types = $ V $
Switchboard phone conversations	2.4 million	20 thousand
Shakespeare	884,000	31 thousand
Google N-grams	1 trillion	13 million (that appear > 40 times)

Corpus (plur. Corpora)

- Corpus: a computer-readable collection of texts
 - E.g., Brown Corpus – a million words large collection of samples from 500 written English texts of different genres (news, fiction, academic, etc.)
- In NLP we often process various corpora
 - Usually the larger, the better as then they are more representative for various linguistic phenomena
- Corpora vary by language/dialect, genre, author demographics, etc.
 - over 6k-7k recognized languages in the world

Vocabulary Growth: Heaps Law

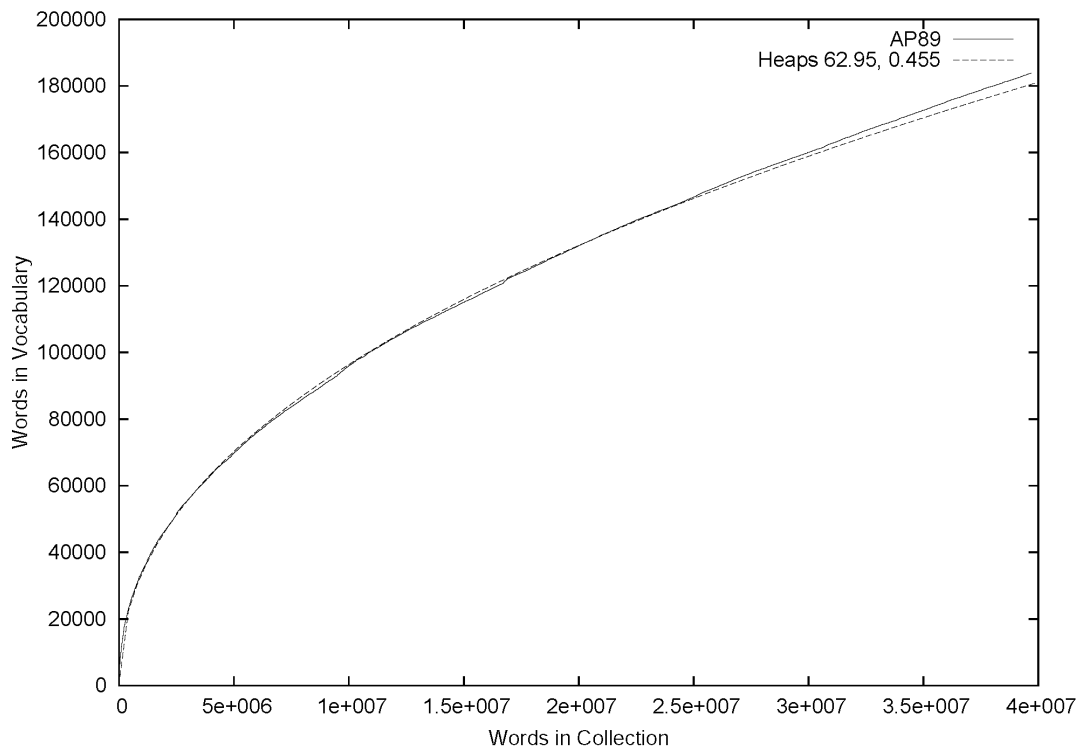
- As document collection grows, so does the size of its vocabulary (total number of different words)
 - Fewer new words are found when collection is already large
- Observed relationship (*Heaps' Law*):

$$|V| = k * N^{\beta}$$

where $|V|$ = vocabulary size (the number of unique words),
 N = total number of words in document collection,
 k, β = parameters that vary for each document collection

(typical values are $10 \leq k \leq 100$ and $\beta \approx 0.5$)

AP89 Example: the Associated Press collection of news stories from 1989

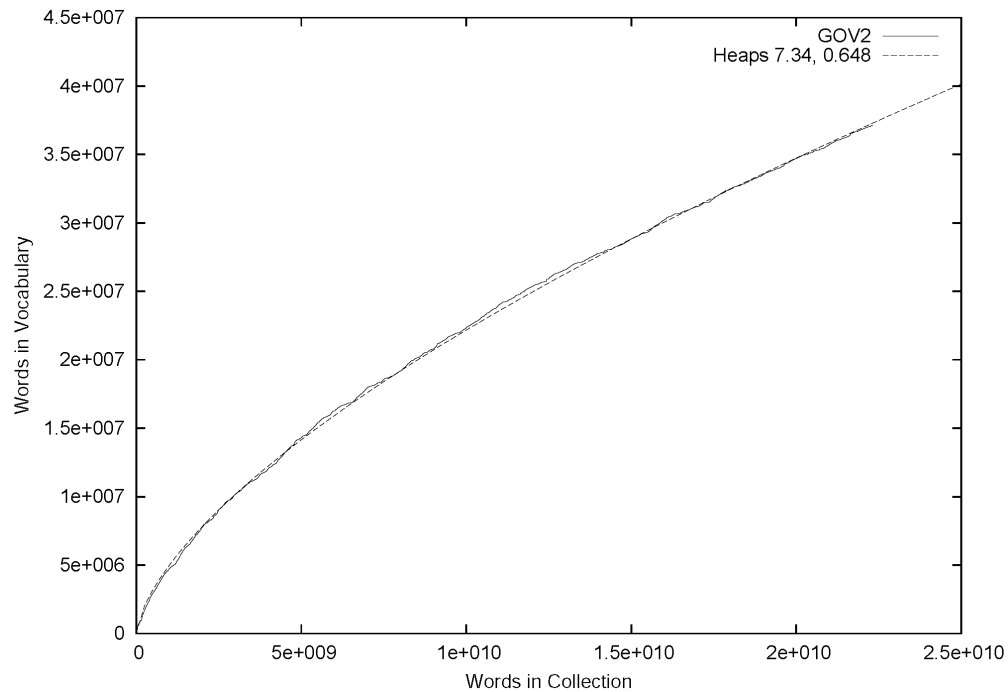


Total documents	84,678
Total word occurrences	39,749,179
Vocabulary size	198,763
Words occurring > 1,000 times	4,169
Words occurring once	70,064

Heaps' Law Predictions

- Predictions for TREC collections are accurate for large numbers of words
 - e.g., first 10,879,522 words of the AP89 collection scanned
 - prediction is 100,151 unique words
 - actual number is 100,024
- Predictions for small numbers of words (i.e. $N < 1000$) are much worse

GOV2 (Web) Example



Web Example

- Heaps' Law works with very large document collections
 - new words occurring even after already seeing 30 million words!
- New words come from a variety of sources
 - spelling errors, invented words (e.g. product, company names), code, other languages, email addresses, etc.
- Search engines must deal with these large and growing vocabularies

Tokenization

- What is a word?
 - A word is any sequence of alphabetical characters between whitespaces that is not a punctuation mark..
- Later we'll ask more questions about words, e.g.:
 - How can we identify different word classes (parts of speech)?
 - What is the meaning of words?
 - How can we represent that?

Simple Tokenization in UNIX

- Naïve tokenization algorithm
- Given a text file, output the word tokens and their frequencies

```
tr -sc 'A-Za-z' '\n' < shakes.txt  
    | sort  
    | uniq -c
```

Change all non-alpha to newlines

Sort in alphabetical order

Merge and count each type

```
1945 A  
  72 AARON  
  19 ABBESS  
   5 ABBOT  
... ..
```

The first step: tokenizing

```
tr -sc 'A-Za-z' '\n' < shakes.txt | head
```

THE

SONNETS

by

William

Shakespeare

From

fairest

creatures

We

...

The second step: sorting

```
tr -sc 'A-Za-z' '\n' < shakes.txt | sort | head
```

A

A

A

A

A

A

A

A

A

...

Counting

- Merging upper and lower case

```
tr 'A-Z' 'a-z' < shakes.txt | tr -sc 'A-Za-z' '\n' | sort |  
uniq -c
```

- Sorting by the counts

```
tr 'A-Z' 'a-z' < shakes.txt | tr -sc 'A-Za-z' '\n' | sort | uniq -c |  
sort -n -r
```

```
23243 the  
22225 i  
18618 and  
16339 to  
15687 of  
12780 a  
12163 you  
10839 my  
10005 in  
8954 d
```

Why this one?

I'd: I had or I would or I should..

Word Frequency

- What we have actually obtained in the previous slide is a frequency distribution of words in Shakespeare texts
- **Word frequency:** the number of occurrences of a word type in a text (or in a collection of texts)
- You may have heard statements such as “adults know about 30,000 words”, “you need to know at least 5,000 words to be fluent”
- Such statements do not refer to inflected word forms (take/takes/taking/take/takes/took) but to lemmas or dictionary forms (take), and assume if you know a lemma, you know all its inflected forms too

Zipf's Law

- How many words occur once, twice, 100 times, 1000 times?
- Zipf's law:
 - **rank (r) of a word multiplied by its frequency (f) is approximately constant (k)**
 - assuming words are ranked in the order of decreasing frequency
 - $r * f \approx k$
 - or
 - $r * P_r \approx c$
 - P_r is probability of word occurrence, and $c \approx 0.1$ for English

News Collection (AP89) Statistics

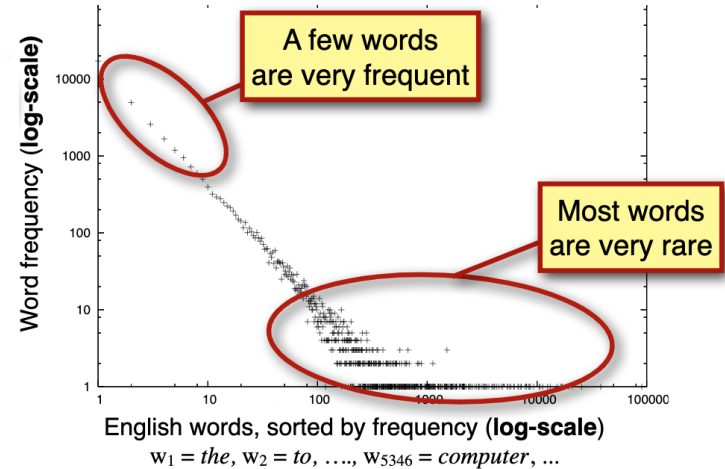
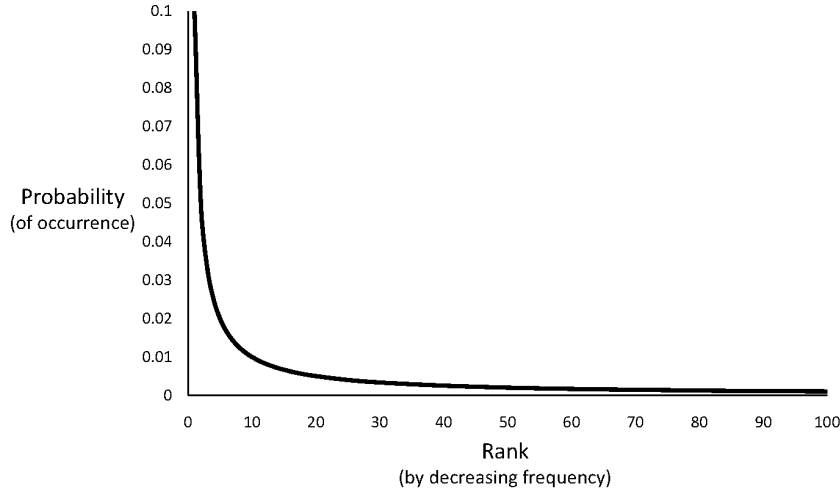
<i>Word</i>	<i>Freq.</i>	<i>r</i>	<i>Pr(%)</i>	<i>r*Pr</i>
assistant	5,095	1,021	.013	0.13
sewers	100	17,110	2.56×10^{-4}	0.04
toothbrush	10	51,555	2.56×10^{-5}	0.01
hazmat	1	166,945	2.56×10^{-6}	0.04

Top 50 Words from AP89

<i>Word</i>	<i>Freq.</i>	<i>r</i>	<i>P_r(%)</i>	<i>r.P_r</i>	<i>Word</i>	<i>Freq.</i>	<i>r</i>	<i>P_r(%)</i>	<i>r.P_r</i>
the	2,420,778	1	6.49	0.065	has	136,007	26	0.37	0.095
of	1,045,733	2	2.80	0.056	are	130,322	27	0.35	0.094
to	968,882	3	2.60	0.078	not	127,493	28	0.34	0.096
a	892,429	4	2.39	0.096	who	116,364	29	0.31	0.090
and	865,644	5	2.32	0.120	they	111,024	30	0.30	0.089
in	847,825	6	2.27	0.140	its	111,021	31	0.30	0.092
said	504,593	7	1.35	0.095	had	103,943	32	0.28	0.089
for	363,865	8	0.98	0.078	will	102,949	33	0.28	0.091
that	347,072	9	0.93	0.084	would	99,503	34	0.27	0.091
was	293,027	10	0.79	0.079	about	92,983	35	0.25	0.087
on	291,947	11	0.78	0.086	i	92,005	36	0.25	0.089
he	250,919	12	0.67	0.081	been	88,786	37	0.24	0.088
is	245,843	13	0.65	0.086	this	87,286	38	0.23	0.089
with	223,846	14	0.60	0.084	their	84,638	39	0.23	0.089
at	210,064	15	0.56	0.085	new	83,449	40	0.22	0.090
by	209,586	16	0.56	0.090	or	81,796	41	0.22	0.090
it	195,621	17	0.52	0.089	which	80,385	42	0.22	0.091
from	189,451	18	0.51	0.091	we	80,245	43	0.22	0.093
as	181,714	19	0.49	0.093	more	76,388	44	0.21	0.090
be	157,300	20	0.42	0.084	after	75,165	45	0.20	0.091
were	153,913	21	0.41	0.087	us	72,045	46	0.19	0.089
an	152,576	22	0.41	0.090	percent	71,956	47	0.19	0.091
have	149,749	23	0.40	0.092	up	71,082	48	0.19	0.092
his	142,285	24	0.38	0.092	one	70,266	49	0.19	0.092
but	140,880	25	0.38	0.094	people	68,988	50	0.19	0.093

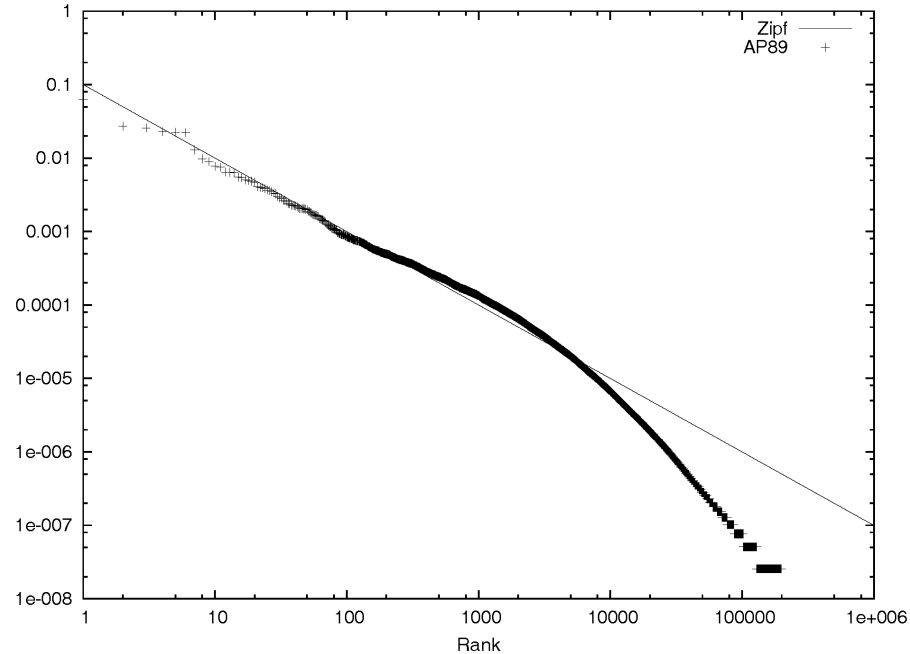
The two most common words (“the”, “of”) make up about 10% of all word occurrences in text documents

Zipf's Law: Probability of Word vs. its Rank



- A small number of events (e.g. words) occur with high frequency (mostly closed-class words like the, be, to, of, and, a, in, that,...)
- A large number of events occur with very low frequency (all open class words)

Zipf's Law for AP89



Some “problems” at very high and low frequencies

Implications of Zipf's Law

- Good News:
 - **Stop words** (commonly occurring words such as “the”, “a”) will account for a large fraction of text so eliminating them greatly **reduces size of vocabulary** of any text
 - We have seen these words often enough that we know (almost) everything about them. These words will help us get at the structure (and possibly meaning) of text
- Bad News:
 - For most words, **gathering sufficient data for meaningful statistical analysis** (e.g. for correlation analysis for query expansion) is difficult since they are extremely rare.
 - We know something about these words, but haven't seen them often enough to know everything about them. They may occur with a meaning or a part of speech we haven't seen before
 - Any text may contain a number of words that are unknown to us. We have never seen them before, but we still need to get at the structure (and meaning) of these texts

Issues in Tokenization

- Finland's capital → Finland Finlands Finland's ?
- what're, I'm, isn't → What are, I am, is not
- Hewlett-Packard → Hewlett Packard ?
- state-of-the-art → state of the art ?
- Lowercase → lower-case lowercase lower case ?
- m.p.h., Ph.D. → ??
- \$45.55 → ?
- 01/02/19 → ?
- #ACL → ?

Words aren't just
defined by blanks..

Issues in Tokenization

- Hong Kong → “Hong”, “Kong” or “Hong Kong”
 - New York-based → ?
 - rock ‘n’ roll → ?
-
- The above examples require multiword expression dictionary
 - Tokenization is also tied with named entity recognition that we will study later

Informal spelling, emoticons, hashtags..



Brianna @_parsimonia_ · 24h

Goooooooood Vibesssssss



@JOYUS · 1m

When idc, I really don't care.

Like my "I want space" is me shutting you out. My "**imma** go, u want something?" And u don't say nothing, then I'm not coming back sumn 4 u



Roger Federer ✓ @rogerfederer · Feb 28

🇧🇪 🇮🇹 🕶 🎬 📷 🎥 🎞 🎧 #Oscars



Dwayne Johnson ✓ @TheRock · Oct 21

And remember to eat pancakes. 💪 🍷 😂



KATY PERRY ✓ @katyperry · 23 Oct 2015

Glad the witch hunt is coming to an end and we can refocus on the future of 🇺🇸

I admire your unbreakable strength

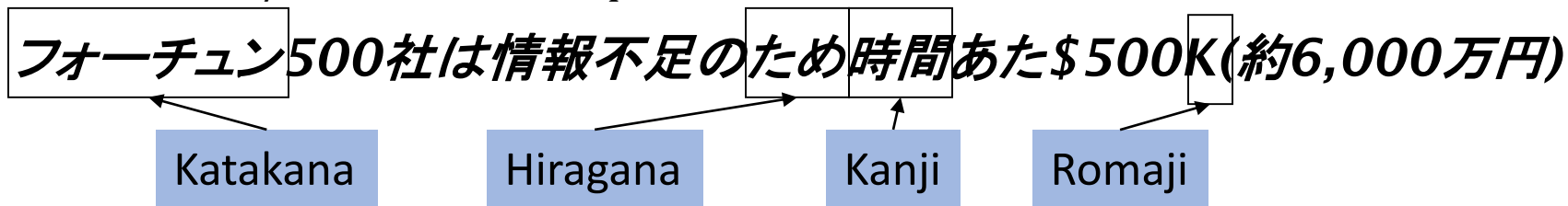
@HillaryClinton #youaboss

Tokenization: language issues

- French (clitic contractions marked by apostrophes)
 - *L'ensemble* → one token or two?
 - *L ? L' ? Le ?*
 - Want *l'ensemble* to match with *un ensemble*
- German noun compounds are not segmented
 - *Lebensversicherungsgesellschaftsangestellter*
 - 'life insurance company employee'
 - German text processing and information retrieval usually needs **compound splitter**

Tokenization: language issues

- Chinese and Japanese languages have no spaces between words:
 - 莎拉波娃现在居住在美国东南部的佛罗里达。
 - 莎拉波娃 现在 居住 在 美国 东南部 的 佛罗里达
 - Sharapova now lives in US southeastern Florida
- Further complicated in Japanese, with multiple alphabets mixed in a sentence
 - Also dates/amounts in multiple formats



End-user could express query entirely in hiragana!

Word Tokenization in Chinese

- Also called **Word Segmentation**
- Chinese words are composed of characters
 - Characters are generally 1 syllable
 - Each character generally represents a single unit of meaning (called a morpheme) and is pronounceable as a single syllable
 - Average word is 2.4 characters long
- Standard baseline segmentation algorithm:
 - Maximum Matching (aka. greedy longest-match-first decoding (MaxMatch))

Maximum Matching Word Segmentation Algorithm

Given a wordlist of Chinese, and a string:

- 1) Start a pointer at the beginning of the string
- 2) Find the longest word in dictionary that matches the string starting at pointer
- 3) Move the pointer over the word in string
- 4) Go to 2

Max-match Segmentation Illustration

- Thecatinthehat
- Thetabledownthere



the cat in the hat
the table down there
theta bled own there

- Doesn't generally work in English..
- But works astonishingly well in Chinese
 - 莎拉波娃现在居住在美国东南部的佛罗里达。
 - 莎拉波娃 现在 居住 在 美国 东南部 的 佛罗里达
- Modern probabilistic or NN-based segmentation algorithms work even better

Character as basic input in Chinese

- In fact, for most Chinese NLP tasks it turns out to work better to take **characters rather than words** as input, since characters are at a reasonable semantic level for most applications
- However, for Japanese and Thai the character is too small a unit, and so algorithms for word segmentation are required

Tokenization Standards

- Any actual NLP system will assume a particular tokenization standard
- Because so much NLP is based on systems that are trained on particular corpora (text datasets) that everybody uses, these corpora often define a de facto standard
- Penn Treebank 3 standard:
 - 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 _" _.

Good practice: be aware of, and better write down, any normalization (tokenization, lowercasing, spell-checking, ...) steps that your system does

Basic Text Processing

Word normalization and stemming

Normalization

- We often need to “normalize” terms
 - E.g. in Information Retrieval indexed text and query terms should have same form
 - For instance, we want to match ***U.S.A.*** and ***USA*** and ***US***

Case folding

- Applications like IR: reduce all letters to lower case
 - Since users tend to use lower case
 - Possible exception: upper case in mid-sentence?
 - e.g., *General Motors*
 - *Fed* vs. *fed*
 - *SAIL* vs. *sail*
- For sentiment analysis, machine translation, information extraction
 - Case is helpful (*US* versus *us* is important) so case folding is usually not done

Lemmatization

- Reduce inflections or variant forms to a base form
 - *am, are, is* → *be*
 - *car, cars, car's, cars'* → *car*
- *the boy's cars are different colors* → *the boy car be different color*
- **Lemmatization**: finding correct dictionary headword form
- Especially important for highly inflected and morphologically complex languages (and useful for machine translation)
 - Spanish **quiero** ('I want'), **quieres** ('you want') have the same lemma as **querer** 'want'

Morphology

- Morphemes:
 - The smallest meaningful units that make up words
- Many word forms consist of a stem plus a number of affixes (prefixes or suffixes)
 - **Stems**: core meaning-bearing units - central morphemes which are usually free morphemes (free morphemes can occur by themselves as words)
 - **Affixes**: providing additional information (usually bound morphemes; bound morphemes have to combine with others to form words)
 - Often with grammatical functions

disgracefully
<prefix><stem><suffix><suffix>

Stemming

- Morphological parsing can be complex so cruder methods are often used
- Reduce terms to their stems (often used in IR)
- *Stemming* is **crude** chopping of affixes
 - language dependent
 - e.g., *automate(s)*, *automatic*, *automation* are all reduced to *automat*

*for example compressed
and compression are both
accepted as equivalent to
compress.*



for exampl compress and
compress ar both accept
as equival to compress

Porter's algorithm: The most common English stemmer: http://9ol.es/porter_js_demo.html

Step 1a

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

Step 1b

(*v*)ing	→	∅	walking	→	walk
			sing	→	sing
(*v*)ed	→	∅	plastered	→	plaster
...					

Step 2 (for long stems)

ational	→	ate	relational	→	relate
izer	→	ize	digitizer	→	digitize
ator	→	ate	operator	→	operate
...					

Step 3 (for longer stems)

al	→	∅	revival	→	reviv
able	→	∅	adjustable	→	adjust
ate	→	∅	activate	→	activ
...					

Based on series of rewrite rules

Porter Stemmer errors

False positives

organization/organ
generalization/generic
numerical/numerous
policy/police
university/universe
addition/additive
negligible/negligent
execute/executive
past/paste
ignore/ignorant
special/specialized
head/heading

False negatives

european/europe
cylinder/cylindrical
matrices/matrix
urgency/urgent
create/creation
analysis/analyses
useful/usefully
noise/noisy
decompose/decomposition
sparse/sparsity
resolve/resolution
triangle/triangular

Dealing with complex morphology is sometimes necessary

- Some languages requires complex morpheme segmentation
 - Turkish
 - **Uygarlastiramadiklarimizdanmissinizcasina**
 - '(behaving) as if you are among those whom we could not civilize'
 - **Uygar + las + tir + ama + dik + lar + imiz + dan + mis + siniz + casina**

Stopping

- Function words (determiners, prepositions) have little meaning on their own
- High occurrence frequencies
- Treated as *stopwords* (i.e. removed)
 - reduce index space, improve response time, improve effectiveness
- Can be important in combinations
 - e.g., “to be or not to be”

Stopping

- Stopword list can be created from high-frequency words or based on a standard list
- Lists are customized for applications, domains, and even parts of documents
 - e.g., “click” is a good stopword for anchor text

a	ourselves
about	out
above	over
after	own
again	same
against	shan't
all	she
am	she'd
an	she'll
and	she's
any	should
are	shouldn't
aren't	so
as	some
at	such
be	than
because	that
been	that's
before	the
being	their
below	theirs
between	them
both	themselves
but	then
by	there
can't	there's
cannot	these
could	they
couldn't	they'd
did	they'll
didn't	they're
do	they've
does	this
doesn't	those
doing	through
don't	to
down	too
during	under
each	until
few	up
for	very
from	was
further	wasn't

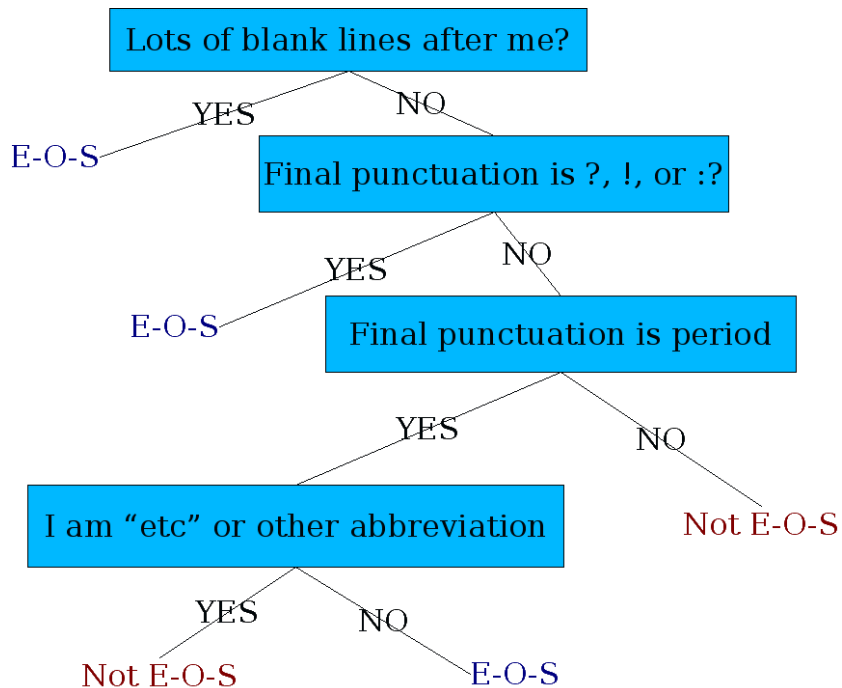
Basic Text Processing

Sentence segmentation

Sentence Segmentation

- !, ? are relatively unambiguous
- Period “.” is quite ambiguous
 - Sentence boundary
 - Abbreviations like Inc. or Dr.
 - Numbers like .02% or 4.3
- Build a binary classifier
 - Looks at a “.”
 - Decides EndOfSentence/NotEndOfSentence
 - Classifiers: hand-written rules, regular expressions, or machine-learning

Determining if a word is end-of-sentence: a simple Decision Tree



E.g., Stanford CoreNLP
has rule-based
sentence splitter

More sophisticated decision tree features

- Case of word with “.”: Upper, Lower, Cap, Number
- Case of word after “.”: Upper, Lower, Cap, Number
- Numeric features
 - Length of word with “.”
 - Probability(word with “.” occurs at end-of-sentence)
 - Probability(word after “.” occurs at beginning-of-sentence)

Decision Trees and other classifiers

- We can think of the questions in a decision tree as features that could be exploited by any kind of classifier
 - Logistic regression
 - SVM
 - Neural Networks
 - etc.

Basic Text Processing

HTML cleaning and preprocessing

Detecting Duplicates

- Duplicate and near-duplicate documents occur in many situations
 - Copies, versions, plagiarism, spam, mirror sites
 - About 30% of the web pages in a large crawl are exact or near duplicates of pages in the other 70%..
- Duplicates consume significant resources during crawling, indexing, and search
 - Yet they present little value to most users

Duplicate Detection

- *Exact* duplicate detection is relatively easy
- *Checksum* techniques
 - A checksum is a value that is computed based on the content of a document
 - e.g., sum of the bytes in the document file
- Possible for files with different text to have the same checksum
- Functions such as a *cyclic redundancy check* (CRC), have been developed that consider the positions of the bytes

T	r	o	p	i	c	a	l		f	i	s	h	<i>Sum</i>
54	72	6F	70	69	63	61	6C	20	66	69	73	68	508

Near-Duplicate Detection

- *Exact* duplicate detection is relatively easy
- Near-duplicate detection - more challenging task
 - E.g., are web pages with the same text content but different advertising or format near-duplicates?
- A near-duplicate document is defined using a threshold value for some similarity measure between pairs of documents
 - e.g., document $D1$ is a near-duplicate of document $D2$ if more than 90% of the words in the documents are the same



Removing Noise

- Many web pages contain text, links, and pictures not directly related to the main content of pages
- This additional material is mostly *noise* that could negatively affect the ranking of the page
- Techniques have been developed to detect the content blocks in a web page
 - Non-content material is either ignored or reduced in importance in the indexing process

Noise Example

CNN.com Member Center Sign In / Register International Edition

SEARCH

Home Page World U.S. Weather Business Sports Analysis Politics Law Technology Science & Space Health Entertainment Offbeat Travel Education Special Reports Video Audio IReports

UPDATE VIDEO 1 EVALUATION

SERVICES E-mail RSS Podcasts Mobile CNN Pipeline

SEARCH

SCIENCE & SPACE

Aquarium plays whale shark matchmaker

Two females flown 8,000 miles for double date in Atlanta

Monday, June 5, 2006, Posted 5:28 p.m. EDT (11:28 GMT)

ATLANTA, Georgia (CNN) — Ralph and Norton, meet Alice and Trase.

The Georgia Aquarium's two male whale sharks got some female companionship on Saturday, when they were joined by two females transported to Atlanta from Taipei, Taiwan.

Researchers are hoping the sharks will mate.

The females — 11 feet and 14 feet long — were flown more than 8,000 miles by UPS, which reconfigured a company 5-747 freighter with advanced marine life support systems to carry them. (Watch what it took to get the sharks together — 1:30)

The pilot said they treated the massive fish the first-class passengers.

"As we were doing the descent, we asked to start down a little sooner to make a nice shallow descent, to not make things too uncomfortable back there for the whale sharks," UPS pilot Capt. Bob Crum said.

The plane's center of balance was carefully planned, according to a statement from the aquarium, and veterinarians accompanied the sharks.

The delivery company also brought the two males to Atlanta, where researchers can study the whale sharks' behavior, breeding and development.

The whale sharks — named after the main characters in the 1950s sitcom "The Honeymooners" — were delivered to the aquarium in special transportation containers.

The Georgia Aquarium, which opened in November, is the world's largest aquarium. It was a \$250 million gift to Georgia from Bernie Marcus, co-founder of The Home Depot and his wife, Bill, through the Marcus Foundation.

It is the only aquarium outside of Asia to showcase whale sharks, which are the largest fish on Earth.

The aquarium's 6.2-million gallon "Ocean Voyager" tank can hold up to six whale sharks at one time.

Image:

YOUR E-MAIL ALERTS

Atlanta (Georgia)

or

[Manage Alerts](#) | [What Is This?](#)

Save up to 75% on Last-Minute Cruises.

Macaroni To Go! Best Price Guarantee GO!

Story Tools

Subscribe to Time for \$3.99

SPACE

TOP STORIES

Astronauts prepare for third spacewalk **Russians choose Putin's successor**

- Astronomers vie to make biggest telescope
- NASA to beam Beatles song to North Star
- U.S. plans for falling satellite
- Iran's president makes landmark visit to Iraq
- Secret PM: Attacks on militants go on
- Cable arrested in abandoned baby case

International Edition Languages CNN TV CNN International Headline News Transcripts Advertise with Us About Us

SEARCH

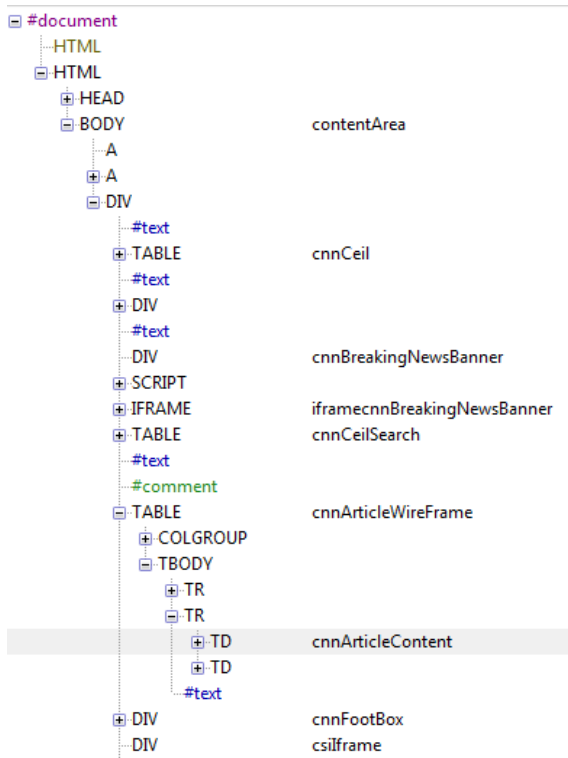
© 2007 Cable News Network. A Time Warner Company. All Rights Reserved. Terms under which this service is provided to you. Read our privacy policies. Contact Us Site Map

External sites open in new window; not endorsed by CNN.com. Pay service with live and archived video. Learn more. Download audio news. Add RSS headlines.

Content
block

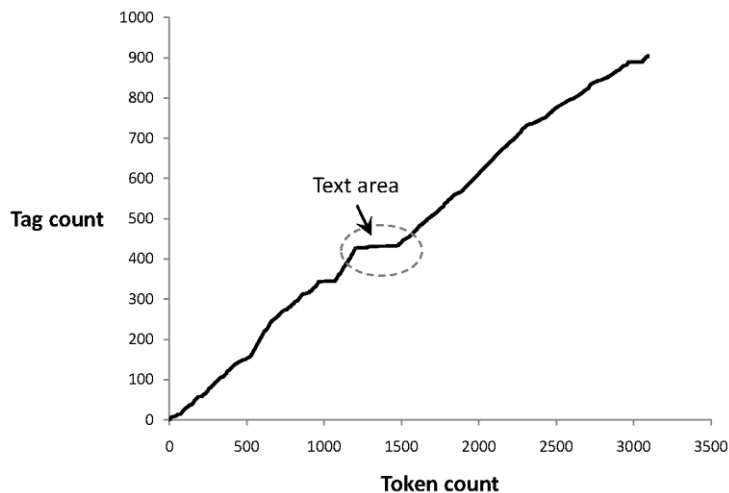
Finding Content Blocks

- Some approaches use DOM structure and visual (layout) features



Finding Content Blocks

- Cumulative distribution of tags in the example web page



- Main text content of the page corresponds to the “plateau” in the middle of the distribution

Finding Content Blocks

- Represent a web page as a sequence of bits, where $b_n = 1$ indicates that the n -th token is a tag
- Optimization problem where we find values of i and j to maximize both the number of tags below i and above j and the number of non-tag tokens between i and j
- i.e., maximize

$$\sum_{n=0}^{i-1} b_n + \sum_{n=i}^j (1 - b_n) + \sum_{n=j+1}^{N-1} b_n$$

Basic Text Processing

Byte Pair Encoding tokenization

Subword segmentation

- **Subword tokenization** (because tokens are often parts of words)
- Data-driven approach (not grounded in morphosyntactic theory)
- Can include common morphemes like *-est* or *-er*
 - A morpheme is the smallest meaning-bearing unit of a language (*unlikeliest* has morphemes *un-*, *likely*, and *-est*)
- Can encode rare and unknown words as sequences of subword units

Subword tokenization

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

Byte Pair Encoding (BPE)

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 training corpus (say 'A', 'B'),
 - adds a new merged symbol 'AB' to the vocabulary
 - replace every adjacent 'A' 'B' in corpus with 'AB'.
- Until k merges have been done.

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)

- Most subword algorithms are run inside white-space separated tokens
- So first add a special end-of-word symbol '_' before whitespace in training corpus
- Next, separate into letters.

BPE token learner

Suppose there is a simple corpus as follows:

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

Add end-of-word tokens and segment:

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

BPE token learner

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
```

Merge **e r** to **er**

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, er
```

BPE

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, er

Merge **er _** to **er_**

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, er, er

BPE

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

Merge **n e** to **ne**

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

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

- On the test data, run all merges learned from the training data:
 - Greedily
 - In the order we learned them
 - (test frequencies don't play a role now)
- So: merge every **e r** to **er**, then merge **er _** to **er_**, etc.
- Result:
 - Test set's "n e w e r _" would be tokenized as a full word
 - Test set's "l o w e r _" (unseen word!) would be two tokens: "low er_"

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

Basic Text Processing

Minimum Edit Distance

Applications of computing string similarity

Spell correction

- The user typed “graffe”

Which is closest?

- graf
- graft
- grail
- giraffe

Computational biology

- Align two sequences of nucleotides

```
AGGCTATCACCTGACCTCCAGGCCGATGCCC  
TAGCTATCACGACCGCGGTTCGATTGCCCGAC
```

- Resulting alignment:

```
-AGGCTATCACCTGACCTCCAGGCCGA--TGCCC---  
TAG-CTATCAC--GACCGC--GGTCGATTGCCCGAC
```

Also useful for machine translation, information extraction, speech recognition, etc.

Edit Distance

- The minimum edit distance between two strings is the minimum number of editing operations:
 - Insertion
 - Deletion
 - Substitution
- which are needed to transform one string into the other

Minimum Edit Distance

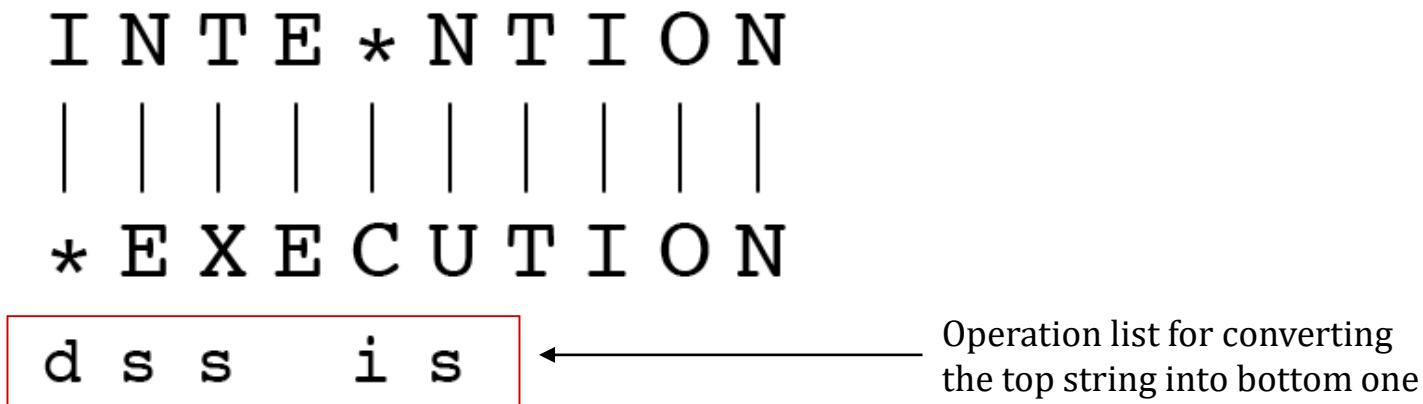
- Two strings and their **alignment**:

I	N	T	E	*	N	T	I	O	N
*	E	X	E	C	U	T	I	O	N

An alignment is a correspondence between
substrings of two sequences

Minimum Edit Distance

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



Alignment in Computational Biology

- Given a sequence of bases

AGGCTATCACCTGACCTCCAGGCCGATGCCC
TAGCTATCACGACCGCGGGTCGATTTGCCCGAC

- An alignment:

–**AG**GCTATCAC**CT**GAC**CT**CC**AG**GC**CGA**––**TGCCC**––
T**AG**–**CT**ATCAC––**GAC**CGC––**GG**T**CGA**TT**TGCCC**GAC

- Given two sequences, align each letter to a letter or gap

Other uses of Edit Distance in NLP

- Evaluating Machine Translation and speech recognition

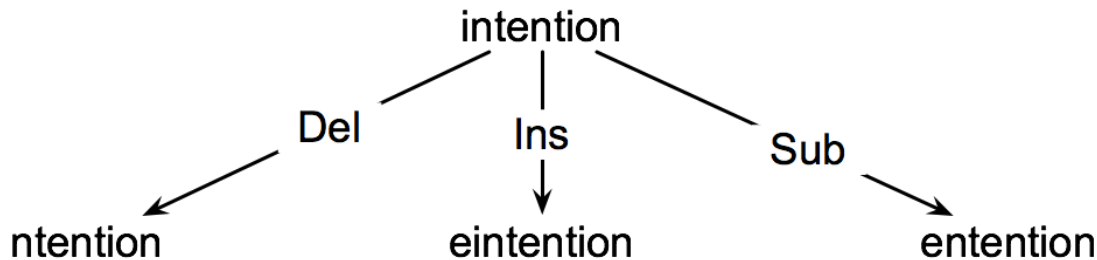
Spokesman confirms senior government adviser was shot
Spokesman said the senior adviser was shot
dead

I S I D

- Named entity extraction and entity coreference
 - IBM Inc. announced today
 - IBM profits
 - Stanford President John Hennessy announced yesterday
 - for Stanford University President John Hennessy

How to find the Min 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 arrive at the same state
 - We don't have to keep track of all of them
 - Just the shortest path to each of those revisited states

i n t e n t i o n
← delete *i*
n t e n t i o n
← substitute *n* by *e*
e t e n t i o n
← substitute *t* by *x*
e x e n t i o n
← insert *u*
e x e n u t i o n
← substitute *n* by *c*
e x e c u t i o n

Path from *intention*
to *execution*

Defining Min Edit Distance

- 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 $D(n,m)$

Dynamic Programming for Minimum Edit Distance

- **Dynamic programming:** A tabular computation of $D(n,m)$
- Solving problems by combining solutions to subproblems
- Bottom-up
 - We compute $D(i,j)$ for small i,j
 - And compute larger $D(i,j)$ based on the previously computed smaller values
 - i.e., compute $D(i,j)$ for all i ($0 < i < n$) and j ($0 < j < m$)

Defining Min Edit Distance (Levenshtein)

- Initialization of $D(\text{source substring}, \text{target substring})$

$$D(i, 0) = i$$

$$D(0, j) = j$$

- Recurrence Relation:

For each $i = 1 \dots M$

For each $j = 1 \dots N$

$$D(i, j) = \min \begin{cases} D(i-1, j) + 1 & \text{deletion} \\ D(i, j-1) + 1 & \text{insertion} \\ D(i-1, j-1) + \begin{cases} 2; & \text{if } X(i) \neq Y(j) \\ 0; & \text{if } X(i) = Y(j) \end{cases} & \text{substitution} \end{cases}$$

- Termination:

$D(N, M)$ is distance

The Edit Distance Table

$$D(i, 0) = i$$

$$D(0, j) = j$$

source

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

target

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

source

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



target

source

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

target

Minimum Edit Distance

Backtrace for Computing Alignments

Computing alignments

- Edit distance isn't sufficient
 - We often need to **align** each character of the two strings to each other
- We do this by keeping a “backtrace”
- Every time we enter a cell, remember where we came from
- When we reach the end,
 - Trace back the path from the upper right corner to read off the alignment

Minimum Edit Distance with Backtrace

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	

The grey colored cells show one possible minimum distance alignment

Adding Backtrace to Minimum Edit Distance

- Base conditions:

$$D(i, 0) = i$$

$$D(0, j) = j$$

- Termination:

$D(N, M)$ is distance

- Recurrence Relation:

For each $i = 1 \dots M$

For each $j = 1 \dots N$

$$D(i, j) = \min \begin{cases} D(i-1, j) + 1 & \text{deletion} \\ D(i, j-1) + 1 & \text{insertion} \\ D(i-1, j-1) + \begin{cases} 2; & \text{if } X(i) \neq Y(j) \\ 0; & \text{if } X(i) = Y(j) \end{cases} & \text{substitution} \end{cases}$$

$$\text{ptr}(i, j) = \begin{cases} \text{LEFT} & \text{insertion} \\ \text{DOWN} & \text{deletion} \\ \text{DIAG} & \text{substitution} \end{cases}$$

Result of Backtrace

- Two strings and their **alignment**:

I N T E * N T I O N
 | | | | | | | | |
 * E X E C U T I O N
 d s s i s

$$\text{ptr}(i, j) = \begin{cases} \text{LEFT} & \text{insertion} \\ \text{DOWN} & \text{deletion} \\ \text{DIAG} & \text{substitution} \end{cases}$$

source

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	

target

Minimum Edit Distance

Weighted Minimum Edit Distance

Weighted Edit Distance

- Why would we add weights to the computation?
 - Spell correction: some letters are more likely to be mistyped than others
 - Biology: certain kinds of deletions or insertions are more likely than others

Confusion matrix for spelling errors (keyboard typing)

sub[X, Y] = Substitution of X (incorrect) for Y (correct)

X	Y (correct)																									
	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
a	0	0	7	1	342	0	0	2	118	0	1	0	0	3	76	0	0	1	35	9	9	0	1	0	5	0
b	0	0	9	9	2	2	3	1	0	0	0	5	11	5	0	10	0	0	2	1	0	0	8	0	0	0
c	6	5	0	16	0	9	5	0	0	0	1	0	7	9	1	10	2	5	39	40	1	3	7	1	1	0
d	1	10	13	0	12	0	5	5	0	0	2	3	7	3	0	1	0	43	30	22	0	0	4	0	2	0
e	388	0	3	11	0	2	2	0	89	0	0	3	0	5	93	0	0	14	12	6	15	0	1	0	18	0
f	0	15	0	3	1	0	5	2	0	0	0	3	4	1	0	0	0	6	4	12	0	0	2	0	0	0
g	4	1	11	11	9	2	0	0	0	1	1	3	0	0	2	1	3	5	13	21	0	0	1	0	3	0
h	1	8	0	3	0	0	0	0	0	0	2	0	12	14	2	3	0	3	1	11	0	0	2	0	0	0
i	103	0	0	0	146	0	1	0	0	0	0	6	0	0	49	0	0	0	2	1	47	0	2	1	15	0
j	0	1	1	9	0	0	1	0	0	0	0	2	1	0	0	0	0	5	0	0	0	0	0	0	0	0
k	1	2	8	4	1	1	2	5	0	0	0	0	5	0	2	0	0	0	6	0	0	0	4	0	0	3
l	2	10	1	4	0	4	5	6	13	0	1	0	0	14	2	5	0	11	10	2	0	0	0	0	0	0
m	1	3	7	8	0	2	0	6	0	0	4	4	0	180	0	6	0	0	9	15	13	3	2	2	3	0
n	2	7	6	5	3	0	1	19	1	0	4	35	78	0	0	7	0	28	5	7	0	0	1	2	0	2
o	91	1	1	3	116	0	0	0	25	0	2	0	0	0	0	14	0	2	4	14	39	0	0	0	18	0
p	0	11	1	2	0	6	5	0	2	9	0	2	7	6	15	0	0	1	3	6	0	4	1	0	0	0
q	0	0	1	0	0	27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
r	0	14	0	30	12	2	2	8	2	0	5	8	4	20	1	14	0	0	12	22	4	0	0	1	0	0
s	11	8	27	33	35	4	0	1	0	1	0	27	0	6	1	7	0	14	0	15	0	0	5	3	20	1
t	3	4	9	42	7	5	19	5	0	1	0	14	9	5	5	6	0	11	37	0	0	2	19	0	7	6
u	20	0	0	0	44	0	0	0	64	0	0	0	0	2	43	0	0	4	0	0	0	0	2	0	8	0
v	0	0	7	0	0	3	0	0	0	0	0	1	0	0	1	0	0	0	8	3	0	0	0	0	0	0
w	2	2	1	0	1	0	0	2	0	0	1	0	0	0	0	7	0	6	3	3	1	0	0	0	0	0
x	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	9	0	0	0	0	0	0	0
y	0	0	2	0	15	0	1	7	15	0	0	0	2	0	6	1	0	7	36	8	5	0	0	1	0	0
z	0	0	0	7	0	0	0	0	0	0	0	7	5	0	0	0	0	2	21	3	0	0	0	0	3	0

