NUS National University of Singapore | School of Computing

Words

CS4248 Natural Language Processing

Week 02

Min-Yen KAN

2

Many slides borrowed with permission from Dan Jurafsky (Stanford) and Hwee Tou Ng (NUS)

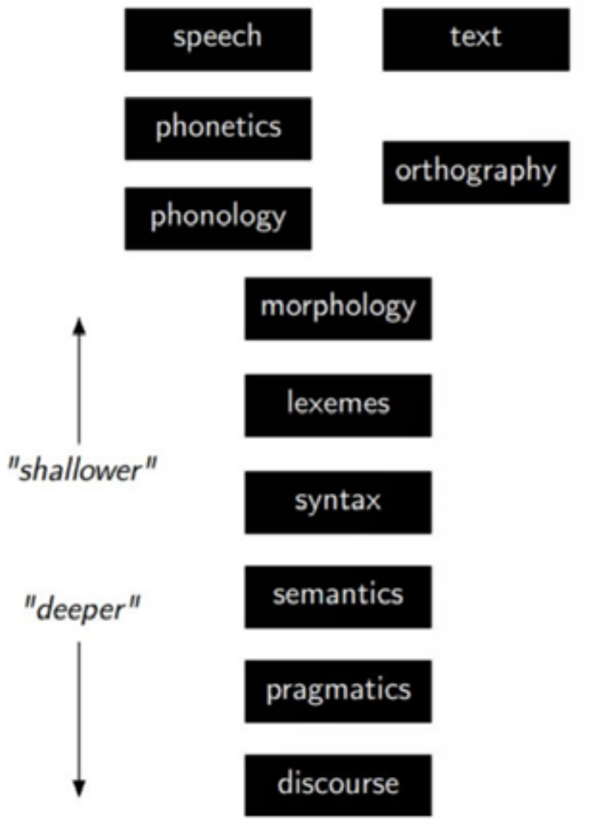
Recap

Levels of Linguistic Knowledge

Why is NLP Hard?

One answer: Ambiguity

We saw the woman with
the telescope wrapped in paper.



Week 02 Agenda

Regular Expressions

Corpus Preprocessing: Getting to Words

- *Detour*: Morphology / Byte Pair Encoding

Normalization

Spelling Errors

Noisy Channel

Edit Distance

Regular Expressions

Slides adapted from Dan Jurafsky (Stanford), Hwee Tou Ng (NUS)

Regular Expressions

A formal language for specifying a set of text strings.

REs can be considered as a pattern to specify text search strings to search a corpus of text.

Show the exact part of the string that first matches the RE pattern.

Slides adapted from Hwee Tou Ng (NUS)

The woodchuck

How can we search for any of these?

- *woodchuck*
- *woodchucks*
- *Woodchuck*
- *Woodchucks*

Photo by Abigail Lynn on Unsplash. Slides adapted from Dan Jurafsky (Stanford)

Regular Expressions: Disjunctions

Letters inside square brackets [ ]

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

Ranges [A-Z]

Pattern	Matches	
`[A-Z]`	An upper case letter	*<u>D</u>renched Blossoms*
`[a-z]`	A lower case letter	*<u>my</u> beans were impatient*
`[0-9]`	A single digit	*Chapter <u>1</u>: Down the Rabbit Hole*

Slides adapted from Dan Jurafsky (Stanford)

Regular Expressions: Negation in Disjunction

Negations [^Ss]: Carat denotes negation only when first in []

Pattern	Matches	
`[^A-Z]`	Not an uppercase letter	*Oyfn pripetchik*
`[^Ss]`	Neither 'S' nor 's'	*I have no exquisite reason"*
`[^e^]`	Neither e nor ^	*Look here*
`a^b`	The pattern a carat b	*Look up a^b now*

Slides adapted from Dan Jurafsky (Stanford)

Regular Expressions: More Disjunctions

Woodchucks is another name for *groundhog*!

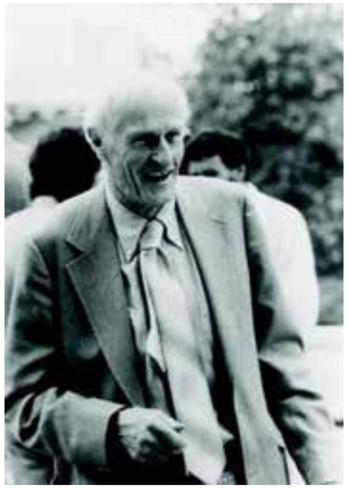
The pipe | for disjunction

Pattern	Matches
`groundhog|woodchuck`	
`yours|mine`	*yours* *mine*
`a|b|c`	= [abc]
`[gG]roundhog|[Ww]oodchuck`	

Photo by Abigail Lynn on Unsplash. Slides adapted from Dan Jurafsky (Stanford)

Regular Expressions: ? * + .

Pattern	Matches	
colou?r	Optional previous char	color colour
oo*h!	0 or more of previous char	oh! ooh! oooh! ooooh!
o+h!	1 or more of previous char	oh! ooh! oooh! ooooh!
baa+		baa baaa baaaa baaaaa
beg.n	Any char	begin begun begun beg3n



Stephen C Kleene

Kleene *, Kleene +

Parenthesis: capture groups.
non-capture groups: (?:)

Regular Expressions: Anchors ^ $

Pattern	Matches
^[A-Z]	Palo Alto
^[^A-Za-z]	1 "Hello"
\.$ fullstop	The end.
.$ wildcard	The end? The end!

( ) precedence
{n} limit exactly n occurences, {n,m} n to m occurences.
{n,} at least n.

Slides adapted from Dan Jurafsky (Stanford)

parenthesis counters

sequences & anchors. disjunction

operator precedence: ( ), * + ? { }, ^ $ sequences, |

b a c ✓ (context free)

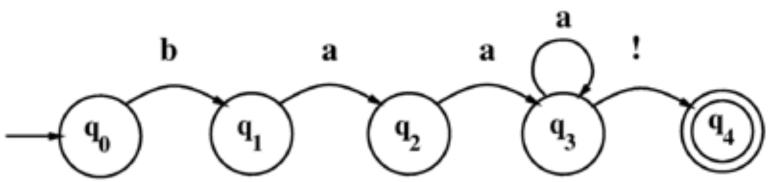
d a e ✗

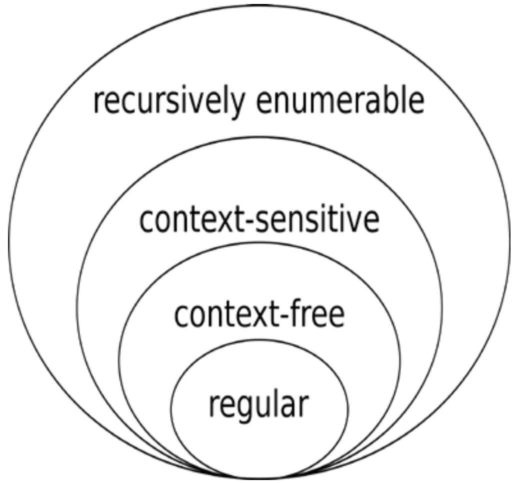
context sensitive

Aside: Why is it called a "Regular" Expression?

Equivalence among Regex, Regular Languages and Finite State Automata (FSA). A regex is an equivalent to an FSA.

context free is more expressive





RE: /baa+!/

baa!
baaa!
baaaa!
baaaaa!
...

	Input		
State	b	a	!
0	1	∅	∅
1	∅	2	∅
2	∅	3	∅
3	∅	3	4
4:	∅	∅	∅

state-transition table

Slides adapted from Hwee Tou Ng (NUS). Picture from Wikipedia.

RE Example and Error Types

Slides adapted from Dan Jurafsky (Stanford), Hwee Tou Ng (NUS)

Regex Example

Find me all instances of the word "*the*" in a text.

[^0-9a-zA-Z][Tt]he[^0-9a-zA-Z]

Slides adapted from Dan Jurafsky (Stanford)

Regex Example

Find me 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]`	numbers?

Slides adapted from Dan Jurafsky (Stanford)

Errors

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

1.

2.

Slides adapted from Dan Jurafsky (Stanford)

Errors

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

1. Matching strings that we should not have matched (*th*er*e*, *then*, o*ther*): False positives (Type I)
2. Not matching things that we should have matched (*The*): False negatives (Type II)

Slides adapted from Dan Jurafsky (Stanford)

Precision = $\frac{TP}{TP + FP}$ Recall = $\frac{TP}{TP + FN}$

Errors

We often deal with these two kinds of errors.

Reducing error then involves two opposing efforts:

- Increasing **accuracy** or **precision** (minimizing false positives)
- Increasing **coverage** or **recall** (minimizing false negatives)

Actual

	+ve	-ve
+ve	TP	FP
-ve	FN	TN

Prediction

Slides adapted from Dan Jurafsky (Stanford)

Summary

Regular expressions can play a surprisingly large role

- Sophisticated sequences of regular expressions are often a key step in text processing
- Outsized role due to cascading effects

For many hard tasks, we use machine learning classifiers

- But regular expressions can provide features in classifiers
- Useful in capturing generalizations

Slides adapted from Dan Jurafsky (Stanford)

Corpus Preprocessing

Every NLP tasks needs to do the prep…

- Tokenization
- *Detour*: Morphology and BPE
- Normalization
- Stemming
- Segmentation

split words by space, etc.

handle OOV

reduce words to standard form (lematization)

only strip suffix from end.

split by sentence

Slides adapted from Dan Jurafsky (Stanford)

How many words?

I do uh main- mainly business data processing

- Fragments, filled pauses

Seuss's cat in the hat is different from other cats!

- **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

Slides adapted from Dan Jurafsky (Stanford)

How many words?

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

Type: an element of the vocabulary.

Token: an instance of that type in running text.

Quick Question: How many types and tokens?

- Tokens?
- Types?

Slides adapted from Dan Jurafsky (Stanford)

How many words?

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

Type: an element of the vocabulary.

Token: an instance of that type in running text.

Quick Question: How many types and tokens?

- 15 Tokens? (or 14)
- 13 Types? (or 12) (or 11?)

Slides adapted from Dan Jurafsky (Stanford)

How many words?

N = number of tokens

V = vocabulary = set of types

$|V|$ is the size of the vocabulary
Church and Gale (1990): $|V| > O(N^{½})$

	Tokens = N	**Types =** $\|V\|$
Switchboard phone conversations	2.4 million	20 thousand
Shakespeare	884,000	31 thousand
Google n-grams	1 trillion	13 million

Slides adapted from Dan Jurafsky (Stanford)

A Tokenization Pipeline

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        25 Aaron
  72 AARON     6 Abate
  19 ABBESS    1 Abates
   5 ABBOT     5 Abbess
 ...         ...
```

Slides adapted from Dan Jurafsky (Stanford)

Tokenizing

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

```
THE
SONNETS
by
William
Shakespeare
From
fairest
creatures
We
...
```

Slides adapted from Dan Jurafsky (Stanford)

Sorting

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

```
A
A
A
A
A
A
A
A
A
...
```

Slides adapted from Dan Jurafsky (Stanford)

Counting

Merging upper and lower case

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

Sorting the counts

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

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

What happened here?

Slides adapted from Dan Jurafsky (Stanford)

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 ?
San Francisco	→	one token or two?
m.p.h., PhD.	→	??

Slides adapted from Dan Jurafsky (Stanford)

Tokenization: language issues

French

- ***L'ensemble*** → one token or two?
  - ***L*** ? ***L'*** ? ***Le*** ?
  - Want ***l'ensemble*** to match with ***un ensemble***

German noun compounds

- ***Lebensversicherungsgesellschaftsangestellter***
  ≡ 'life insurance company employee'
- German needs **compound splitter**

Slides adapted from Dan Jurafsky (Stanford)

Tokenization: language issues

Chinese and Japanese 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'

Multiple syllabaries are intermingled in Japanese

- Also dates/amounts can be in multiple formats

フォーチュン500社は情報不足のため時間あた$500K(約6,000万円)

Katakana Hiragana Kanji Romanji

End-user can express query entirely in Hiragana!

Slides adapted from Dan Jurafsky (Stanford)

Word Tokenization in Chinese

Also called Word Segmentation

Chinese words are composed of characters

- Characters are generally 1 syllable and 1 morpheme.
- Average word is 2.4 characters long.

Standard baseline segmentation algorithm:

- Maximum Matching (also called Greedy)

Slides adapted from Dan Jurafsky (Stanford)

Maximum Matching

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

Slides adapted from Dan Jurafsky (Stanford)

Max-match segmentation illustration

Thecatinthehat the cat in the hat

Thetabledownthere 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 segmentation algorithms even better

Slides adapted from Dan Jurafsky (Stanford)

Corpus Preprocessing

Morph + olog + ical _ Pro + cess + ing

- Tokenization
- *Detour*: Morphology and BPE
- Normalization
- Stemming
- Segmentation

NUS School of Computing

Slides adapted from Dan Jurafsky (Stanford) and Hwee Tou Ng (NUS)

Morphology

The study of the way words are built up from morphemes

cats = *cat* + *-s*

Morphemes: The minimal meaning-bearing unit in a language ("morph" ≡ shape)

- Stems: The core meaning-bearing units: *cat*
- Affixes: Bits and pieces that adhere to stems. *-s*
  Often carry grammatical function

Photo by The Lucky Neko on Unsplash.. Slides adapted from Dan Jurafsky (Stanford)

Why Morphology?

Listing all the different morphological variants of a word in a dictionary is inefficient.

Affixes are productive; they apply to new words (e.g., *fax* and *faxing*).

For morphologically complex languages like Turkish, it is impossible to list all morphological variants of every word.

Slides adapted from Hwee Tou Ng (NUS)

Forms of Morphology

Inflectional

- Combine a stem and an affix to form a word in the same class as stem
- For syntactic function like agreement
- e.g., *-s* to form plural form of a noun

Derivational

- Combine a stem and an affix to form a word in a different class
- Harder to predict the meaning of the derived form
- e.g., -*ation* in *computerize* and *computerization*

Slides adapted from Hwee Tou Ng (NUS)

Byte Pair Encoding

Getting to Words

- Tokenization
- *Detour*: Morphology and BPE
- Normalization
- Stemming
- Segmentation

Out of Vocabulary (OOV)

New Words are created all of the time.

Manfuckinghattan, *Twitterati*, *kiasuism*

Morphological analysis can apply to them, even when seen for the first time. (know which are nouns, adjectives?.

All gompies are biff and luff voomly.

M'moon is a cramy gompy, she is the biffiest and luffs voomly too.

Repurposed Byte Pair Encoding (BPE)

```
aaabdaaabac

aaabdaaabac    Replace Z = aa

ZabdZabac      Replace Y = ab

ZYdZYac        Replace X = ZY

XdXac          Final compressed string
```

Replacement Table

Byte pair	Replacement
X	ZY
ab	Y
aa	Z

Image from Akashdeep Singh Jaswal @ Medium

BPE Algorithm

Originally a compression algorithm.

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

k = a tunable parameter (corpus size dependent).

Spectrum between word and character tokens (subwords).

Normalization

Getting to Words

- Tokenization
- *Detour*: Morphology and BPE
- **Normalization**
- Stemming
- Segmentation

Slides adapted from Dan Jurafsky (Stanford) and Hwee Tou Ng (NUS)

Normalization

Convert text to a convenient, standard form

- Application to retrieval: indexed text & query terms must have same form. We want to match ***U.S.A.*** and ***USA***

We implicitly define equivalence classes of terms

- e.g., deleting periods in a term

Alternative: asymmetric expansion:

- Enter: ***window*** Search: ***window***, ***windows***
- Enter: ***windows*** Search: ***Windows***, ***windows***, ***window***
- Enter: ***Windows*** Search: ***Windows***

Potentially more powerful, but less efficient

Slides adapted from Dan Jurafsky (Stanford)

Case folding

Fold: Applications like IR: Reduce letters to lowercase

- Since users tend to use lower case
- Possible exception: upper case in mid-sentence?
  - e.g., ***General Motors***
  - ***Fed*** vs. ***fed***
  - ***MOM*** vs. ***mom***

Don’t Fold: Applications like sentiment analysis, MT

- ***US*** versus ***us*** is important

Slides adapted from Dan Jurafsky (Stanford)

Lemmatization

Reduce inflections or variant forms to 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: find the correct dictionary headword form

Application: Machine translation

- Spanish *quiero* ('I want'), *quieres* ('you want'): same lemma as *querer* ('to want')

Slides adapted from Dan Jurafsky (Stanford)

Penn Treebank Tokenization

- Separate out clitics
  - *doesn't → does n't*
  - *John's → John 's*

Clitic: A morpheme that carries the role of an independent word, but cannot stand on its own; only occurs attached to another word.

- Keep hyphenated words together

- Separate out all punctuation symbols

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 " .

Slides adapted from Hwee Tou Ng (NUS)

Corpus Preprocessing

Getting to Words

- Tokenization
- *Detour*: Morphology and BPE
- Normalization
- Stemming
- Segmentation

Slides adapted from Dan Jurafsky (Stanford) and Hwee Tou Ng (NUS)

Stemming

Reduce terms to their stems in information retrieval

Stemming is the crude chopping of affixes

- language dependent
- e.g., *automate(s)*, *automatic*, *automation* 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*

Slides adapted from Dan Jurafsky (Stanford)

Porter Stemmer (uses regex)

https://tartarus.org/martin/PorterStemmer/

A simple and efficient stemming algorithm used in information retrieval

- A series of rewrite rules run in a cascade; the output of each pass is fed as input to the next pass
  - *ational* ⟶ *ate* (e.g., *relational* ⟶ *relate*)
  - *ing* ⟶ ε if stem contains vowel (e.g., *motoring* ⟶ *motor*)
  - *sses* ⟶ *ss* (e.g., *grasses* ⟶ *grass*)

- Does not require a lexicon

Slides adapted from Hwee Tou Ng (NUS)

Dealing with complex morphology

Some languages require complex morpheme segmentation

- Turkish: *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* 'p1pl' + *dan* 'abl'
  + *mis* 'past' + *siniz* '2pl' + *casina* 'as if'

Slides adapted from Dan Jurafsky (Stanford)

Corpus Preprocessing

Words … to Sentences

Detour #2: Features and Classifiers

- Tokenization
- *Detour*: Morphology and BPE
- Normalization
- Stemming
- **Segmentation**

Slides adapted from Dan Jurafsky (Stanford)

Sentence Segmentation

!, *?* are relatively unambiguous

But period "." is quite ambiguous

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

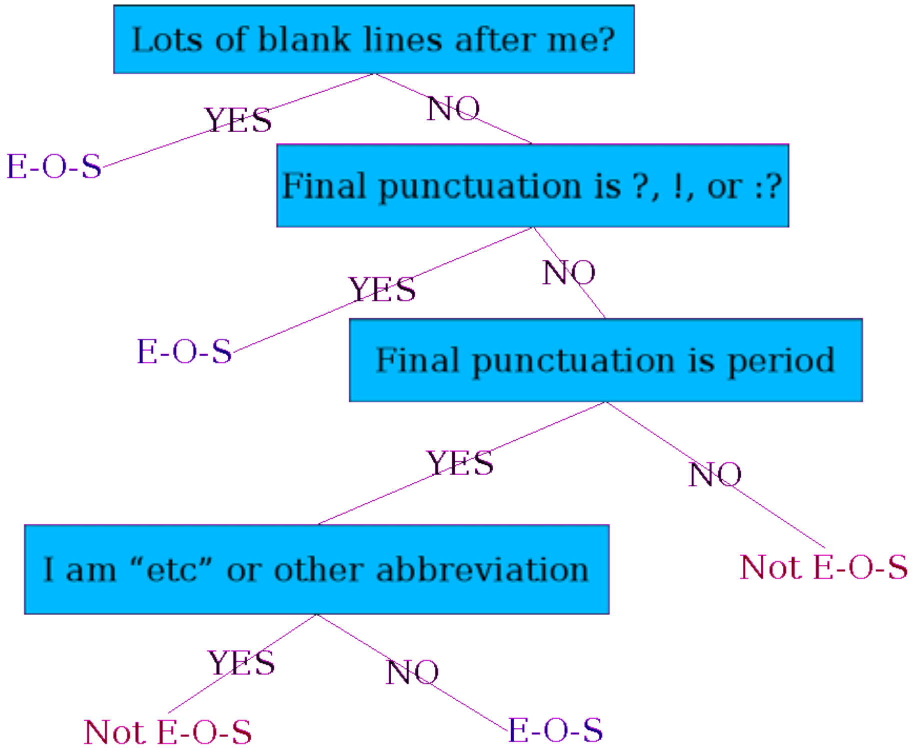
One solution: Build a binary classifier

- Looks at a "."
- Decides EndOfSentence/NotEndOfSentence

Classifiers: hand-written rules, regular expressions, or machine learning

Slides adapted from Dan Jurafsky (Stanford)

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



Slides adapted from Dan Jurafsky (Stanford)

More sophisticated 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-s)
- Probability (word after "." occurs at beginning-of-s)

Key Takeaway: think of the questions in a decision tree as features that could be exploited by any kind of classifier

Slides adapted from Dan Jurafsky (Stanford)

Spellling Errars

http://archive.google.com/jobs/britney.html

Slides adapted from Dan Jurafsky (Stanford)

Spelling Errors

Three increasingly broader problems:

1. Non-word error detection
   - E.g., detecting *graffe* (misspelling of *giraffe*)
2. Isolated-word error correction
   - Consider a word in isolation
   - E.g., correcting *graffe* to *giraffe*
3. Context-sensitive error detection and correction
   - Use of context to detect and correct spelling errors
   - Real-word errors
   - *there* for *three*, *dessert* for *desert*, *piece* for *peace*

Let's Try:

Acress ?

Slides adapted from Hwee Tou Ng (NUS)

Spelling Error Patterns

Most misspelled words in typewritten text are single-error

- Dameru (1964): 80%, Peterson (1986): 93-95%

Single-error misspellings:

- Insertion: mistyping *actress* as *acress*
- Deletion: mistyping *cress* as *acress*
- Substitution: mistyping *access* as *acress*
- Transposition: mistyping *caress* as *acress*

Slides adapted from Hwee Tou Ng (NUS)

Candidate generation

Words with similar spelling

- Small edit distance to error

Words with similar pronunciation

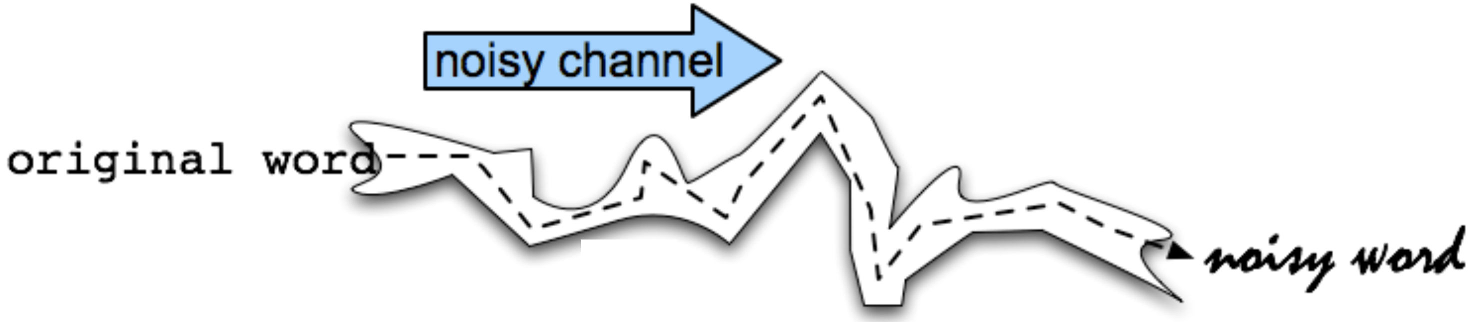
- Small edit distance of pronunciation to error

Slides adapted from Dan Jurafsky (Stanford)

Noisy Channel Model

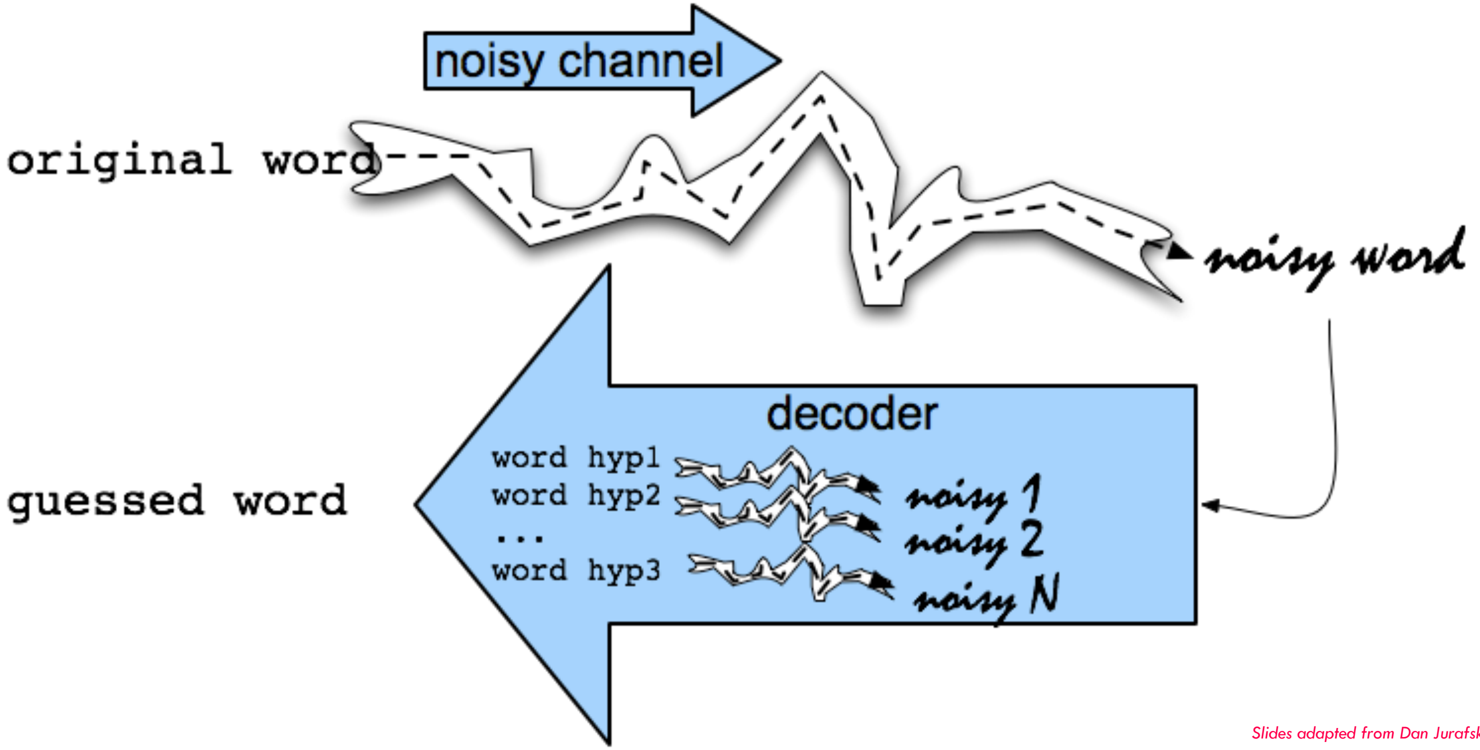
Slides adapted from Dan Jurafsky (Stanford)

Noisy Channel Intuition



Slides adapted from Dan Jurafsky (Stanford)

Noisy Channel Intuition



Slides adapted from Dan Jurafsky (Stanford)

Noisy Channel

We see an observation x of a misspelled word

Find the correct word w:

$$\hat{w} = \underset{w \in V}{\operatorname{argmax}} P(w|x)$$

$$\hat{w} = \underset{w \in V}{\operatorname{argmax}} \frac{P(x|w)P(w)}{P(x)}$$ given, $=1$

$$\hat{w} = \underset{w \in V}{\operatorname{argmax}} P(x|w)|P(w)$$

Slides adapted from Dan Jurafsky (Stanford)

Scoring candidates

Need a corpus of annotated text where misspelled words are identified and labeled with the correctly spelled ones.

Gather the probability estimates from the annotated corpus

Slides adapted from Hwee Tou Ng (NUS)

Estimating the Prior $P(w)$

Use a Maximum Likelihood Estimate (MLE): $P(w) = \frac{freq(w)}{N}$

word	Frequency of word	P(word)
actress	9,321	.0000230573
cress	220	.0000005442
caress	686	.0000016969
access	37,038	.0000916207
across	120,844	.0002989314
acres	12,874	.0000318463

Slides adapted from Hwee Tou Ng (NUS), Dan Jurafsky (Stanford)

del $[x_{i-1}, w_i]$

Estimating the Likelihood $P(x|w)$

ins $[x_{i-1}, w_i]$

count$[w_{i-1}]$

$$P(x|w) = \begin{cases} \frac{\cancel{\text{del}[w_{i-1}, w_i]}}{\text{count}[w_{i-1} w_i]}, & \text{if deletion} \\ \frac{\cancel{\text{ins}[w_{i-1}, x_i]}}{\text{count}[w_{i-1}]}, & \text{if insertion} \\ \frac{\text{sub}[x_i, w_i]}{\text{count}[w_i]}, & \text{if substitution} \\ \frac{\text{trans}[w_i, w_{i+1}]}{\text{count}[w_i w_{i+1}]}, & \text{if transposition} \end{cases}$$

Kernighan, Church, Gale (1990)

del [x, y] : count (xy typed as x)
ins [x, y] : count (x typed as xy)
sub [x, y] : count (x typed as y)
trans [x, y] : count (xy typed as x)

Slides adapted from Hwee Tou Ng (NUS), Dan Jurafsky (Stanford)

Confusion matrix for spelling errors

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

Y (correct)

X	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	0	5	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	0	27	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

Slides adapted from Dan Jurafsky (Stanford)

Channel model for acress

Candidate Correction	Correct Letter	Error Letter	x|w	P(x|word)
actress	t	-	c|ct	.000117
cress	-	a	a|#	.00000144
caress	ca	ac	ac|ca	.00000164
access	c	r	r|c	.000000209
across	o	e	e|o	.0000093
acres	-	s	es|e	.0000321
acres	-	s	ss|s	.0000342

Slides adapted from Hwee Tou Ng (NUS), Dan Jurafsky (Stanford)

Noisy channel probability for acress

Candidate Correction	Correct Letter	Error Letter	x|w	P(x|word)	P(word)	10^9 *P(x|w)P(w)
actress	t	-	c|ct	.000117	.0000231	2.7
cress	-	a	a|#	.00000144	.00000054	.00078
caress	ca	ac	ac|ca	.00000164	.00000170	.0028
access	c	r	r|c	.000000209	.0000916	.019
across	o	e	e|o	.0000093	.000299	2.8
acres	-	s	es|e	.0000321	.0000318	1.0
acres	-	s	ss|s	.0000342	.0000318	1.0

Slides adapted from Hwee Tou Ng (NUS), Dan Jurafsky (Stanford)

Edit Distance

A method for judging word similarity orthographically

Slides adapted from Dan Jurafsky (Stanford)

Candidate generation

80% of errors are within edit distance 1

Almost all errors within edit distance 2

Also allow insertion of space or hyphen

- `thisidea` → `this idea`
- `inlaw` → `in-law`

Other uses of Edit Distance in NLP

- Evaluating Machine Translation and speech recognition

```
R Spokesman confirms    senior government adviser was shot
H Spokesman said    the senior            adviser was shot dead
               S      I             D                        I
```

- Named Entity Extraction and Entity Coreference

  IBM Inc. announced today
  IBM profits
  Stanford President John Hennessy announced yesterday
  for Stanford University President John Hennessy

Slides adapted from Dan Jurafsky (Stanford)

Applications in Computational Biology

Align two sequences of nucleotides

AGGCTATCACCTGACCTCCAGGCCGATGCCC
TAGCTATCACGACCGCGGTCGATTTGCCCGAC

Resulting alignment:

−AGGCTATCACCTGACCTCCAGGCCGA--TGCCC---
TAG-CTATCAC--GACCGC--GGTCGATTTGCCCGAC

- Comparing genes or regions from different species
  - to find important regions
  - determine function
  - uncover evolutionary forces
- Assembling fragments to sequence DNA
- Compare individuals to looking for mutations

Slides adapted from Dan Jurafsky (Stanford)

Edit Distance

The minimum edit distance between two strings.

Allowed edit operations needed to transform one into the other

- Insertion
- Deletion
- Substitution
- Transposition

Not handling this last one for now. Think about how to do it yourself.

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

Slides adapted from Dan Jurafsky (Stanford)

Minimum Edit Distance

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

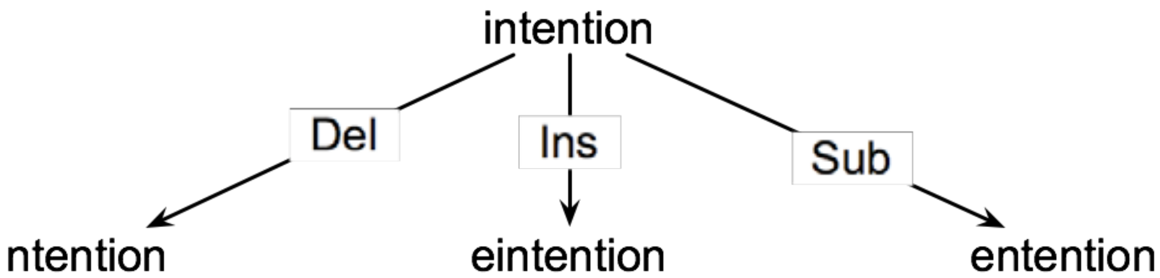
- If each operation has cost of 1, what is the cost?
  Answer:
- If substitutions cost 2 (1 insertion, 1 deletion), cost?
  Answer: 5

Slides adapted from Dan Jurafsky (Stanford)

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



Slides adapted from Dan Jurafsky (Stanford)

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.
  - Key insight: We don't have to keep track of all of them!
  - Just the shortest path to each of those revisited states.

Slides adapted from Dan Jurafsky (Stanford)

Defining Min Edit Distance

For two strings (x of length n & y of length m), define $d(i, j)$:

Why 0 and not 1?

- the edit distance between $x[0..i]$ and $y[0..j]$
  - i.e., the first i characters of x and the first j characters of y
- The edit distance between x and y is thus $d(n, m)$

Slides adapted from Dan Jurafsky (Stanford)

School of Computing

Computing Edit Distance

Dynamic Programming: Not dynamic and not programming

Slides adapted from Dan Jurafsky (Stanford)

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 previously computed smaller values
- i.e., compute $d(i, j)$ for all i ($0 < i \leq n$) and j ($0 < j \leq m$)

Slides adapted from Dan Jurafsky (Stanford)

Defining Min Edit Distance (Levenshtein)

- Initialization

```
d(i,0) = i
d(0,j) = j
```

- Recurrence Relation:

```
For each  i = 1…m
        For each  j = 1…n
```

insertion

substitute

$$d(i,j) = \min \begin{cases} d(i-1,j) + 1 \\ d(i,j-1) + 1 \\ d(i-1,j-1) + \begin{cases} 2; \text{ if } x(i) \neq y(j) \\ 0; \text{ if } x(i) = y(j) \end{cases} \end{cases}$$

match

- Termination:

```
d(n,m) is distance
```

Slides adapted from Dan Jurafsky (Stanford)

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

Slides adapted from Dan Jurafsky (Stanford)

Minimum Edit Distance

N	9									
O	8									
I	7									
T	6									
N	5									
E	4									
T	3									
N	2									
I	1	2	3							
#	0	1	2	3	4	5	6	7	8	9
	#	E	X	E	C	U	T	I	O	N

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

delete "I"
insert "E"

(i-1, j) = delete "I"

steps needed for x[0..i]
to be y[0..j]

Slides adapted from Dan Jurafsky (Stanford)

Minimum Edit Distance – Filled

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

Slides adapted from Dan Jurafsky (Stanford)

Backtrace for Edit Distance

Slides adapted from Dan Jurafsky (Stanford)

Result of a Backtrace

- Two strings and their **alignment:**

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

Slides adapted from Dan Jurafsky (Stanford)

Cost ≠ Alignment

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

Slides adapted from Dan Jurafsky (Stanford)

Min 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**

Slides adapted from Dan Jurafsky (Stanford)

Adding Backtrace to Minimum Edit Distance

- Base conditions:
  `d(i,0) = i` `d(0,j) = j`

Termination:
`d(n,m) is distance`

- Recurrence Relation:

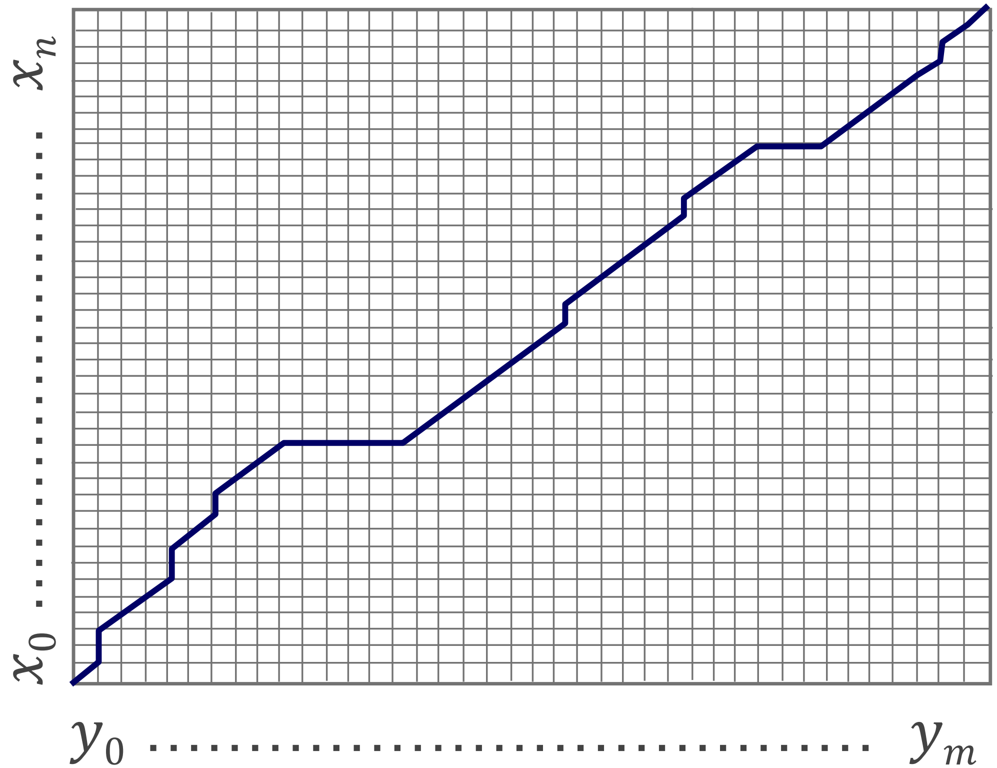
```
For each  i = 1…m
      For each  j = 1…n
```

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

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

Slides adapted from Dan Jurafsky (Stanford)

The Distance Matrix



Every non-decreasing path from $(0,0)$ to (m, n) corresponds to an alignment of the two sequences.

An optimal alignment is composed of optimal subalignments.

Slides adapted from Dan Jurafsky (Stanford)

Weighted Min Edit Distance

Because some edits are more expensive than others

Slides adapted from Dan Jurafsky (Stanford)

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

Slides adapted from Dan Jurafsky (Stanford). Photo from [Clay Banks @ Unsplash.]

Confusion matrix for spelling errors

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	0	5	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	0	27	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

Nice to see you again!

Slides adapted from Dan Jurafsky (Stanford)

Weighted Min Edit Distance

- Initialization:

```
d(0,0) = 0
d(i,0) = d(i-1,0) + del[x(i)];     1 < i ≤ n
d(0,j) = d(0,j-1) + ins[y(j)];     1 < j ≤ m
```

- Recurrence Relation:

```
               ⎧ d(i-1,j)   + del[x(i)]
d(i,j)= min    ⎨ d(i,j-1)   + ins[y(j)]
               ⎩ d(i-1,j-1) + sub[x(i),y(j)]
```

- Termination:

```
d(n,m) is distance
```

Summary: Edit Distance

Slides adapted from Dan Jurafsky (Stanford)

Big O Performance

- Time:
- Space:
- Backtrace:

Slides adapted from Dan Jurafsky (Stanford)

Big O Performance

- Time: $O(nm)$
- Space: $O(nm)$
- Backtrace: $O(n+m)$

Slides adapted from Dan Jurafsky (Stanford)

NUS School of Computing

Variants on the theme

- Needleman-Wunsch
  - OK to have an unlimited # of gaps in the beginning and end
  - If so, we don’t want to penalize gaps at the ends

- Smith-Wasserman
  - Ignore badly aligning regions
  - Find optimal local alignments withing substrings