

CSCI 3907/6907

Fall 2019

Lecture 2 – Part II

Words & Transducers

Text Normalization

- *Corpus* → a computer-readable collection of text or speech
- 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
 4. Morphological analysis

Natural language processing (NLP) is a subfield of linguistics, computer science, information engineering, and artificial intelligence concerned with the interactions between computers and human (natural) languages, in particular how to program computers to process and analyze large amounts of natural language data.

Challenges in natural language processing frequently involve speech recognition, natural language understanding, and natural language generation.

What counts as a word?

Example: *He stepped out into the hall, was delighted to encounter a water brother.*

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

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

- *hall,* is different than *hall*
- *Hall* is different than *hall*
- Sometime punctuation marks count as words (useful for part-of-speech tagging)

What counts as a word?

Example: I do **uh** **main-** mainly business data processing

- In spoken utterances:
 - Fragments (e.g. main-) and filled pauses (e.g. uh or um)

stripped in speech transcripts but might be treated as words in speech recognition (e.g. restarting idea or clause)

Wordform & Lemma

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

- **Wordform**: the full inflected or derived surface form
 - **cat** and **cats** = different wordforms
- **Lemma**: same stem (main word), major part of speech (noun), rough word sense
 - **cat** and **cats** = same lemma **cat**

How many words?

- **Type:** an element of the vocabulary (e.g. *the word cat is counted once*)
- **Token:** an instance of that type in running text (e.g. *the word cat is counted twice*)

How many types and tokens in the following sentence?

They picnicked by the pool, then lay back on the grass and looked at the stars.

- 16 tokens and 14 types

They picnicked by **the** pool, then lay back on **the** grass and looked at **the** stars.

Word Normalization

- Putting words/tokens in a standard format
 - Dates represented in a unified format
 - USA and US → USA
 - uh-huh and uhhuh → uhhuh
- Some information is lost
- Can be valuable for information retrieval or information extraction

Case Folding

- Mapping all upper cases to lower cases
- Helpful for generalization in information retrieval or speech recognition
- Not very helpful for sentiment analysis, information extraction, and machine translation
 - (e.g. US the country or us the pronoun)

Simple Tokenization in UNIX

- Separate the tokens in input text -- whitespaces for now

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

THE

SONNETS

by

William

Shakespeare

From

fairest

creatures

we

...

Simple Tokenization in UNIX – Cont.

- sort the tokens

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

A

A

A

A

A

A

A

A

A

...

Simple Tokenization in UNIX – Cont.

- Merging upper and lower case and output unique instances and their frequencies

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

27597 the

26738 and

22538 i

19771 to

18138 of

14725 a

13826 you

12489 my

11536 that

...

Issues in Tokenization

Word Tokenization (Segmentation) → the process of segmenting running text into words

- 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. → ??

Penn Treebank Tokenization

- Separate out clitics (doesn't becomes does and n't)
- Keep hyphenated words together
- Separate all punctuation

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	”	.

- English Tokenization is typically implemented with FSAs.

Tokenization: Language Issues

- French
 - *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 information retrieval needs **compound splitter**

Tokenization: Language Issues

- Chinese and Japanese 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 intermingled
 - Dates/amounts 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 Romaji

Word Tokenization in Chinese

- 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/max-match (Greedy)

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

illustration

- Thecatinthehat → The cat in the hat
- Thetabledownthere → The table down there
OR
Theta bled own there

Max-match Segmentation

- 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

Morphology

- In natural languages, words are made up of meaningful subunits called **morphemes** (e.g. fox → one morpheme or cats → two morphemes cat and s)
- Morphology → the study of the way words are built up from morphemes
- Morphemes may be
 - **Stems**: the main morpheme of the word
 - **Affixes**: convey the word's role, number, gender, etc.
- Examples
 - cats == cat [stem] + s [suffix]
 - undo == un [prefix] + do [stem]

Morphology

- Important information for some applications
 - part-of-speech tagging
 - spelling correction
 - information retrieval in some languages like Russian (e.g. Moscow has different endings in Moscow, of Moscow, to Moscow ... etc but they mean the same thing)
- English morphology:
 - Inflectional morphology
 - Derivational morphology
 - Cliticization

Inflectional Morphology

- Combination of a stem and affix, results in a word in the same class
 - Example: cat and cats
- Relatively simple in English:
 - Only nouns, verbs, and some adjectives can be inflected
- The number of inflectional affixes is small

Inflectional Morphology - Nouns

- plural affix
 - spelled –s after most nouns, -es after words ending in s, z, sh, ch, or x.
 - Nouns ending in y are changed to –ies (comedy/comedies).

	Regular Nouns		Irregular Nouns	
Singular	cat	thrush	mouse	ox
Plural	cats	thrushes	mice	oxen

- possessive affix
 - apostrophe + -s for regular singular nouns (*e.g. llama's*) and plural nouns not ending in -s (*children's*)
 - a lone apostrophe after regular plural nouns (*llamas'*) and some names ending in -s or -z (*Euripides' comedies*).

Inflectional Morphology – Regular Verbs

- Regular verbs have 4 morphological forms:

Morphological Form Classes	Regularly Inflected Verbs			
stem	walk	merge	try	map
-s form	walks	merges	tries	maps
-ing participle	walking	merging	trying	mapping
Past form or -ed participle	walked	merged	tried	mapped

- The majority of verbs in English are regular.
- Regular verbs are a productive class: automatically includes any new words that enter the language.

Inflectional Morphology – Irregular Verbs

- A much smaller class of verbs

Morphological Form Classes	Irregularly Inflected Verbs		
stem	eat	catch	cut
-s form	eats	catches	cuts
- <i>ing</i> participle	eating	catching	cutting
Past form	ate	caught	cut
- <i>ed</i> participle	eaten	caught	cut

Derivational Morphology

- Combination of a stem and a grammatical morpheme that usually results in a word of a different class.
 - Example: kill (verb) and killer (noun)
- **Nominalization:** forming nouns from verbs or adjectives.

Suffix	Base Verb/Adjective	Derived Noun
-ation	computerize (V)	computerization
-ee	appoint (V)	appointee
-er	kill (V)	killer
-ness	fuzzy (A)	fuzziness

Derivational Morphology – Cont.

- Forming adjectives from nouns or verbs

Suffix	Base Noun/Verb	Derived Adjective
-al	computation (N)	computational
-able	embrace (V)	embraceable
-less	clue (N)	clueless

- Derivation is less productive than inflection.
 - For example, the suffix `-ation` cannot be added to absolutely every verb.

Cliticization

- A **clitic** → a unit whose status lies between an affix and a word
 - Phonologically and orthographically, behaves like affixes (short and unaccented)
 - Syntactically, behaves like words (pronouns, articles, conjunction, or verbs)
- English clitics include these auxiliary verbal forms:
 - am → 'm, are → 're, is → 's, will → 'll, have → 've, has → 's, had → 'd, would → 'd

Cliticization – Cont.

- Clitics in English are ambiguous
 - He's → is or has
 - generally easy to detect with the presence of apostrophes.
- Clitics are harder to parse in other languages (Arabic and Hebrew, for example)

Morphological Parsing

Parsing → taking an input and producing some sort of linguistic structure

Morphological parsing → the problem of recognizing that a word breaks down into morphemes and building a structured representation

Input	Morphological Parsed Output
cats	cat +N +PL
cat	cat +N +SG
cities	city +N +PL
geese	goose +N +PL
goose	(goose +N +SG) or (goose +V)
gooses	goose +V +3SG
merging	merge +V +PRES-PART
caught	(catch +V +PAST-PART) or (catch +V +PAST)

Morphological Parsing

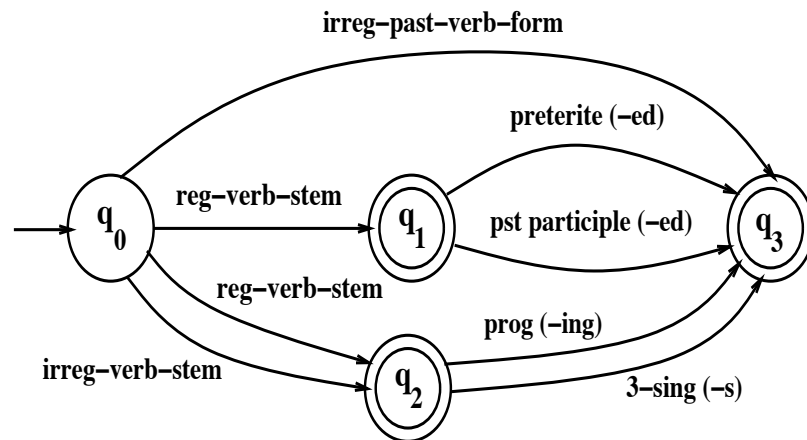
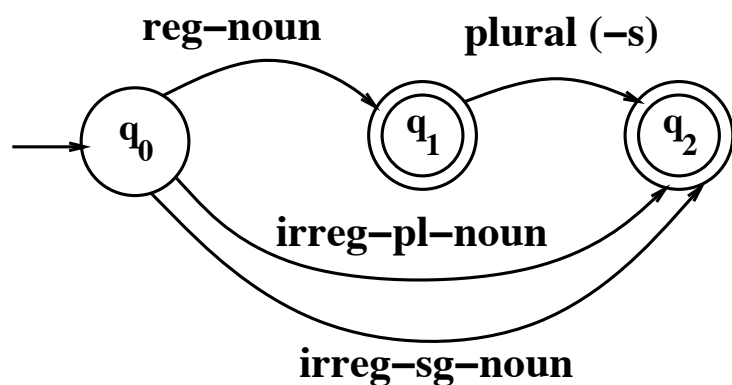
- To build a morphological parser, we need
 - a **lexicon**: The list of stems and affixes, together with basic information about them (whether a stem is a Noun stem or a Verb stem, etc)
 - **morphotactics**: the model of morpheme ordering that explains which classes of morphemes can follow other classes of morphemes inside a word.
 - e.g. English plural morpheme follows the noun rather than preceding it.
 - **orthographic rules**: spelling modifications that may occur when affixation occurs
 - e.g city → cities

Morphological Parsing

- Creating a lexicon
 - Identifying valid words in a language
- Generate words from morphological features
 - catch +V +PAST → caught
- Analyze a word into morphological features
 - caught → catch +V +PAST

Construction of a Finite-State Lexicon

- Since inflectional morphology is productive, it's far more efficient to build a finite-state lexicon instead of a listing of all possible word forms.



A finite-state automaton for English nominal inflection.

Derivational Morphology

- English Derivational morphology is sufficiently complex that FSAs for modeling it can be quite complex.
- Example: Adjectives:

big, bigger, biggest

cool, cooler, coolest, coolly

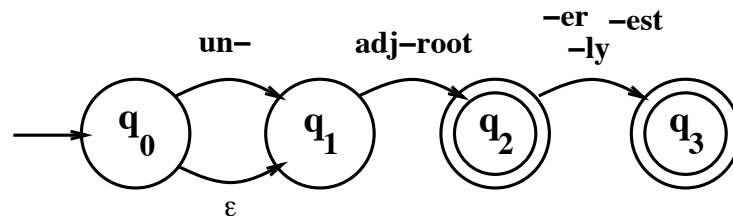
red, redder, reddest

clear, clearer, clearest, clearly, unclear, unclearly

happy, happier, happiest, happily

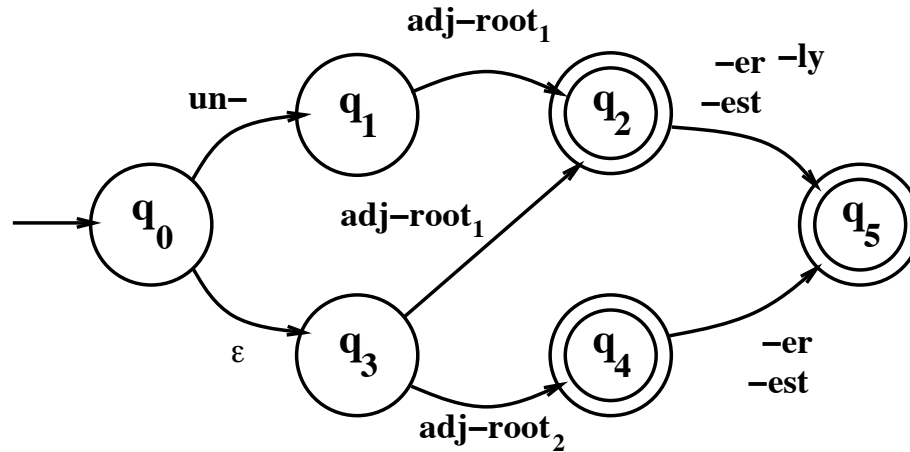
unhappy, unhappier, unhappiest, unhappily

real, unreal, really



False positives: unbig, oranger, smally

A Slightly Better Construction



Finite State Transducers

- Similar to FSAs, but includes output in addition to input
 - Multiple tapes
- Maps between two sets of symbols
- Multiple ways of thinking about transducers:
 - Translator: given an input string, generate an output string.
- For morphological parsing, we use FSTs as translators.

Sequential Transducers

- FSTs that are deterministic on their input.
 - Given an input string, there is one possible output string.
 - Output may contain epsilon symbols (empty string), but not the input.
- In finite-state morphology, a word is represented as a correspondence between a lexical level and a surface level.

Lexical {

	c	a	t	+N	+PL			
--	----------	----------	----------	-----------	------------	--	--	--

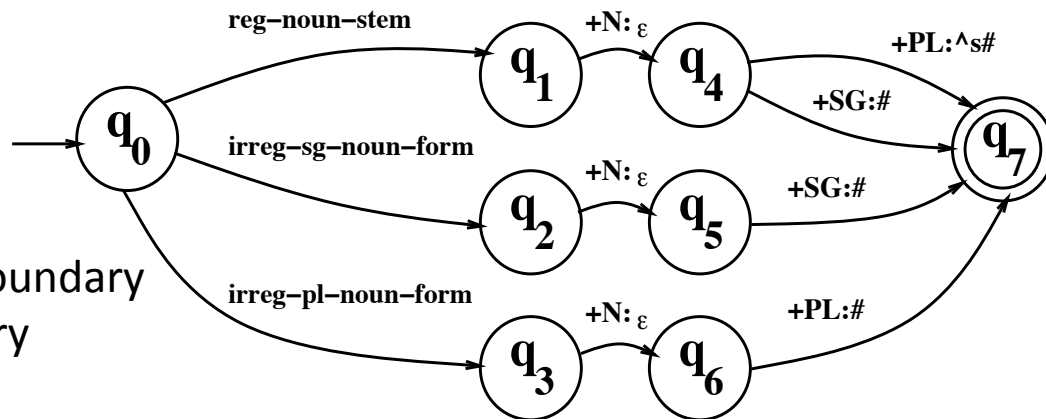
 }

Surface {

	c	a	t	s				
--	----------	----------	----------	----------	--	--	--	--

 }

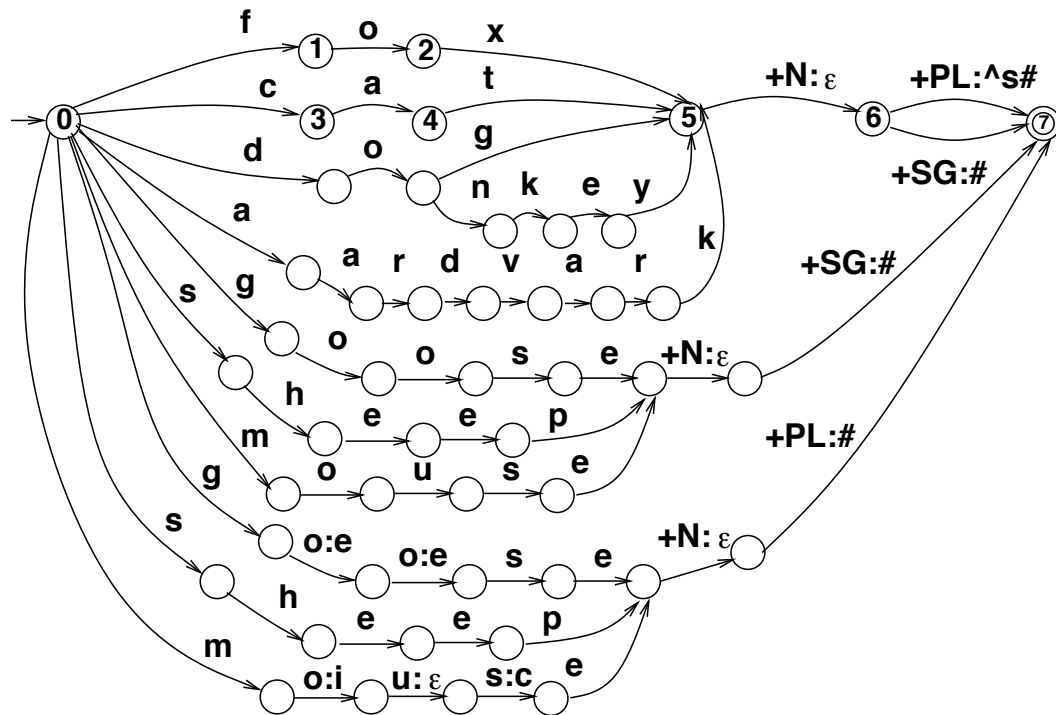
Example: FST for Nominal Inflection



^ indicates a morpheme boundary
indicates a word boundary

reg-noun	irreg-pl-noun	irreg-sg-noun
fox	g o:e o:e s e	goose
cat	sheep	sheep
dog	m o:i u:ε s:c e	mouse
aardvark		

Fleshed-out Nominal Inflection FST

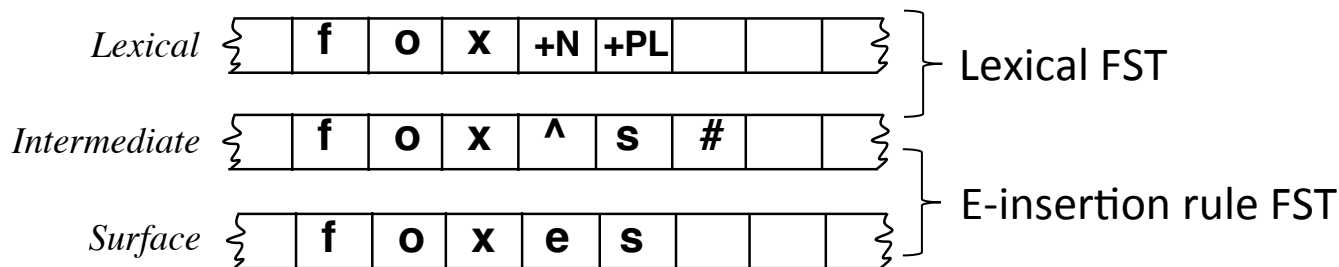


Spelling Rules & FSTs

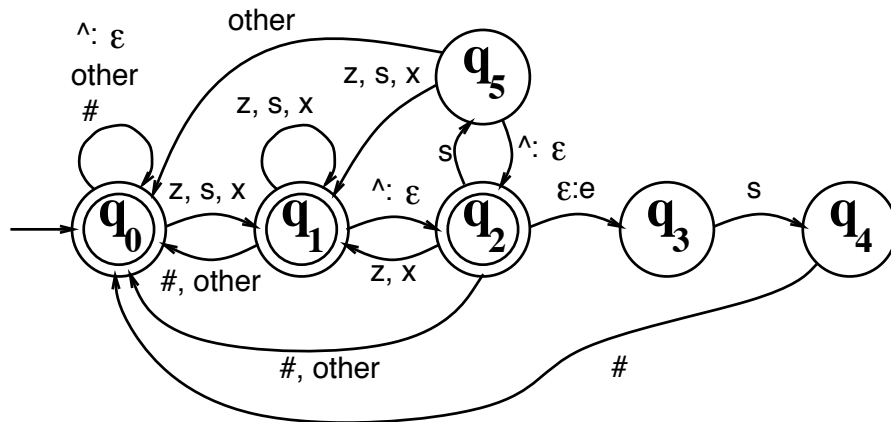
- The above FST would reject “foxes” and accept “foxs”
- To incorporate spelling rules, we can add an intermediate tape (or multiple intermediate tapes if necessary).
- Some spelling rules:

Name	Description of Rule	Example
Consonant doubling	1-letter consonant doubled before <i>-ing/-ed</i>	beg/begging
E deletion	Silent e dropped before <i>-ing</i> and <i>-ed</i>	make/making
E insertion	e added after <i>-s,-z,-x,-ch, -sh</i> before <i>-s</i>	watch/watches
Y replacement	-y changes to <i>-ie</i> before <i>-s</i> , <i>-i</i> before <i>-ed</i>	try/tries
K insertion	verbs ending with <i>vowel + -c</i> add <i>-k</i>	panic/panicked

E-insertion Rule with Intermediate Tape



E-insertion rule Transducer:



Parsing and Disambiguation

- Parsing is more complicated because of ambiguity.
 - For example, “foxes” can be either a verb or a noun
 - “I saw two foxes yesterday” **fox +N +PL**
 - “That trickster foxes me every time!” **fox +V +3Sg**
- Disambiguation typically requires external evidence such as surrounding words.
 - For now, the FST can enumerate all possibilities.

Lemmatization

- Reduce inflections or variant forms to base form
 - *am, are, is* → *be*
 - *car, cars, car's, cars'* → *car*
- Lemmatization: have to find correct dictionary headword form
- Applications:
 - Information Retrieval:
retrieve pages with the word “woodchucks” if we search “woodchuck”

Stemming

- Naïve form of morphological analysis → stemming
- *Stemming* is crude chopping-off affixes
 - language dependent
 - More efficient than full morphological parsing and useful in IR tasks.
 - 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 are both accepted
as equivalent to *compress*

The Porter Stemmer

- One of the most widely used stemming algorithms.
- A series of simple rewrite rules.
- Can be viewed as a lexicon-free FST
- Example rules:

ATIONAL → ATE (e.g. relational → relate)

ING → ε if stem contains vowel (e.g. motoring → motor)

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

How similar are two strings?

- Spell correction

- The user typed “graffe”

Which is closest?

- graf
 - graft
 - grail
 - giraffe

- Also for Machine Translation, Information Extraction, Speech Recognition

Edit Distance

- A mean to quantify these intuitions about string similarity
- The minimum edit distance between two strings
 - the minimum number of editing operations (Insertion, Deletion, and Substitution) needed to transform one into the other
 - Example: graffe → giraffe or grail ?

Minimum Edit Distance

- Two strings and their **alignment**.
- Alignment → a correspondence between substrings of the two sequences

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

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				

- d:deletion s:subtitution i:insertion
- If each operation has cost of 1 → Distance between these is 5
- If substitutions cost 2 (Levenshtein) → Distance between them is 8

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

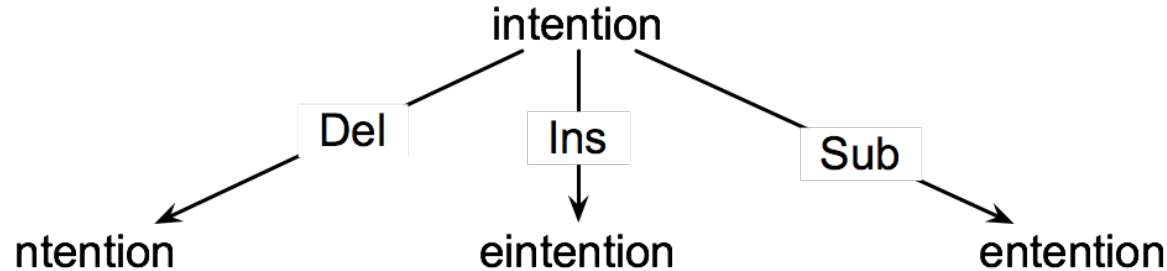
D

I

- Named Entity extraction and entity co-reference
 - 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 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 re-visited states.

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 sub-problems.
- 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 ($0 < i < n$) and j ($0 < j < m$)

Defining Min Edit Distance

- **Initialization**

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

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

- **Recurrence Relation:**

For each $i = 1 \dots M$

For each $j = 1 \dots N$

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


- **Termination:** $D(N, M)$ is distance

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

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

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


The Edit Distance Table

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

The Edit Distance Table

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

The Edit Distance Table

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

The Edit Distance Table

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

The Edit Distance Table

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

Computing alignments

- Edit distance isn't sufficient
 - We often need to **align** each character of the two strings to each other (minimum cost alignment)
- 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

MinEdit 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	

Adding Backtrace to Minimum Edit Distance

- Base conditions:

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

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

Termination:

$$D(N, M) \text{ is distance}$$

- Recurrence Relation:

For each $i = 1 \dots M$

For each $j = 1 \dots N$

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

Performance

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

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

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	

Weighted Min Edit Distance

- Initialization:

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

$$D(i,0) = D(i-1,0) + \text{del}[x(i)]; \quad 1 < i \leq N$$

$$D(0,j) = D(0,j-1) + \text{ins}[y(j)]; \quad 1 < j \leq M$$

- Recurrence Relation:

$$D(i,j) = \min \begin{cases} D(i-1,j) & + \text{del_cost}[x(i)] \\ D(i,j-1) & + \text{ins_cost}[y(j)] \\ D(i-1,j-1) & + \text{sub_cost}[x(i),y(j)] \end{cases}$$

- Termination:

$D(N,M)$ is distance