

Regular Expression

Recap of the last lecture

NLP layers

- Understanding the semantics is a non-trivial task.
- Needs to performs a series of incremental tasks to achieve this.
- NLP happens in layers

Pragmatics & Discourse	<i>Study of semantics in context.</i>
Semantics	<i>Meaning of the sentence.</i>
Parsing	<i>Syntactic structure of the sentence.</i>
Chunking	<i>Grouping of meaningful phrases.</i>
Part of speech tagging	<i>Grammatical classes.</i>
Morphology	<i>Study of word structure.</i>



Increasing
Complexity Of
Processing

Regular Expression

Regular Expression (RE)

- A standard notation of characterizing a text sequence
- How can we search for any of the following:
 - woodchuck
 - woodchucks
 - Woodchuck
 - Woodchucks



- RE search requires a pattern and a **corpus** of texts to search through.

Regular Expression (RE)

RE	Example patterns matched
<code>woodchunks</code>	"interesting links to <u>woodchanks</u> and..."
<code>a</code>	"Ma <u>r</u> y Ann stopped by Mona's"

- RE is case-sensitive
- Letters inside square brackets []

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 impatient
<code>[0-9]</code>	A single digit	Chapter <u>1</u> : Down the Rabbit Hole

Negation

- Negations `[^Ss]`
 - Carat means negation only when it appears immediately after “[“

Pattern	Matches	Example patterns matched
<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>[e^]</code>	Either ‘e’ or ‘^’	Look h <u>e</u> re
<code>a^b</code>	The pattern ‘a’ carat ‘b’	Look up <u>a^b</u> now

It solves the problem of woodchuck vs. Woodchuck

But not woodchuck vs woodchucks

Not woodchuck vs groundhog

Question mark ?

- ? Makes optimality of the pervious expression

Pattern	Matches
Woodchucks?	Woodchuck or Woodchucks
Colou?r	Color or Colour

It solves woodchuck vs woodchucks

Not woodchuck vs groundhog

pipe | for disjunction

Pattern	Matches
<code>groundhog woodchuck</code>	Woodchucks is another name for groundhog
<code>yours mine</code>	yours mine
<code>??a b c</code>	= <code>[abc]</code>
<code>[gG]roundhog [Ww]oodchuck</code>	

It solves woodchuck vs groundhog

But not woodchuckssssssssss

Kleene *, Kleene +

- Kleene * => zero or more occurrences of the immediately previous character or regular expression
- Kleene + => one or more of the previous character
- Period (.) matches any single character (except a carriage return)

Pattern	Matches	
oo*h!	0 or more of previous char	<u>oh!</u> <u>ooh!</u> <u>oooh!</u> <u>ooooh!</u>
[ab]*	Zero or more a's or b's	<u>aaa</u> <u>ababab</u> <u>bbbb</u>
o+h!	1 or more of previous char	<u>oh!</u> <u>ooh!</u> <u>oooh!</u> <u>ooooh!</u>
baa+		<u>baa</u> <u>baaa</u> <u>baaaa</u> <u>baaaaa</u>
beg.n		<u>begin</u> <u>begun</u> <u>begun</u> <u>beg3n</u>

Anchors ^ \$

- Caret ^ matches the start of a line
 - Negations [^Ss] (careful!)
- \b matches a word boundary
- \B matches a non-boundary
- \$ matches the end of a line

Pattern	Matches
<code>^[A-Z]</code>	
<code>^[^A-Za-z]</code>	
<code>\bthe\b</code>	
<code>\. \$</code>	
<code>.\$</code>	

Anchors ^ \$

- Caret ^ matches the start of a line
 - Negations [^Ss] (careful!)
- \b matches a word boundary
- \B matches a non-boundary
- \$ matches the end of a line

Pattern	Matches
<code>^[A-Z]</code>	<u>P</u> alo Alto
<code>^[^A-Za-z]</code>	<u>1</u> <u>"Hello"</u>
<code>\bthe\b</code>	<u>the</u> , not "other"
<code>\. \$</code>	The end <u>.</u>
<code>. \$</code>	The end <u>?</u> The end <u>!</u>

Quiz

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

the

You may miss capitalized examples

[tT]he

Incorrectly returns other or theology

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

It won't find the word the when it begins or ends a line

(^|[^a-zA-Z])[tT]he([a-zA-Z]|\$) ⇒ Before the we require either the beginning-of-line or a non-alphabetic character, and the same at the end of the line.

Advanced Operators

RE	Expansion	Match	Example Patterns
\d	[0-9]	any digit	Party_of_5
\D	[^0-9]	any non-digit	Blue_moon
\w	[a-zA-Z0-9_]	any alphanumeric or space	Daiyu
\W	[^\w]	a non-alphanumeric	!!!!
\s	[_\r\t\n\f]	whitespace (space, tab)	
\S	[^\s]	Non-whitespace	in_Concord

Error

- We want to fix two kinds of errors
 - False positives (Type I)
 - Matching strings that we should not have matched (there, then, other)
 - False negatives (Type II)
 - Not matching things that we should have matched (The)
- Reducing error may require a trade-off between
 - Accuracy or Precision: minimizing false positives
 - Coverage or Recall: minimizing false negative

Morphology

Challenges...

- How do we know that
 - Both *woorchunk* and woodchuncks have same original/root word?
 - May be easy: the plural just tacks as s on to end
 - But what about goose vs geese or fox vs foxes?
- **Two kinds of knowledge:**
 - **Orthographic rules:** can solve *woorchunk* vs. woodchuncks
 - **Morphological rules:** can distinguish goose vs geese

Orthographic/Spelling Rules

- General rules used when breaking a word into its stem and modifiers.
- Example:
 - Singular English words ending with –y, when pluralized, end with –ies.
 - Peccary vs. Peccaries

Morphology: Definition

The study of words, how they are formed, and their relationship to other words in the same language.

Morphological Rules

- Morphological rules are exceptions to the orthographic rules used when breaking a word into its stem and modifiers.
- Example:
 - Goose vs Geese is due to vowel change

Morphological Parsing

- **Parsing:** Take an input and produce some sort of linguistic structure
- Morphological parsing the process of determining the morphemes from which a given word is constructed.

Terminologies

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

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

- **Surface form:** Raw text present in a corpus
 - Example: going
- **Token:** a word present in running text (may be duplicated)
- **Type:** Unique word present in the running text
- **Vocabulary:** Set of types

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

- 15 tokens (or 14)
- 13 types (or 12) (or 11?)

- **Stem**
- **Affix : prefix/suffix/infix/circumfix**

Infix

Tagalog: hingi (borrow) => humingi

Circumfix

German: Sagen (to say) => gesagt (said)

Terminologies

- A word can have more than one affix
 - Example: rewrites (*re-*, *write*, *-s*), unbelievably (*un-*, *believe*, *-able*, *-ly*)
- English doesn't tend to stack more than 4 or 5 affixes
- Languages that tend to string affixes together like Turkish does are called **agglutinative** languages
 - Turkish can have words with 9 or 10 affixes

Terminologies

- **Inflection:** A word stem with a grammatical morpheme, usually resulting in a word of the same class as the original stem.
 - **Example:** Plural (-s) or past (-ed)
- **Derivation:** the combination of a word stem with a grammatical morpheme, usually resulting in a word of a different class
 - **Example:** computerize (verb) vs. computerization (noun)

Compounding: combination of two stems: e.g., *doghouse*

Cliticization: Combination of a word stem with a clitic: e.g., *I've*

Inflectional Morphology

- **Two kinds of inflection:** an affix that marks **plural** and an affix that marks **possessive**

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

- Possessive suffix is realized by
 - apostrophe + -s for regular singular nouns (*llama's*)
 - plural nouns not ending in -s (*children's*)
 - often by a lone apostrophe after regular plural nouns (*llamas'*)

Derivational Morphology

- **Recall:** derivation is the combination of a word stem with a grammatical morpheme, usually resulting in a word of a *different class*.
- **Nominalization:** Forming a new noun from verb/adjective

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

- Adjectives can also be derived from nouns and verbs

Suffix	Base Noun/Verb	Derived Adjective
-al	computation (N)	computational

Concatenative morphology is Easy!

- **Non-concatenative morphology**

- Philipian language (Tagalog)
- Um + hingi (request) = humingi (ask for)

- **Templatic morphology/root-and-pattern morphology**

- In Hebrew, a verb is constructed from two components: a root (CCC) and a template (ordering of C and V to specify more semantic info)
- *lmd* (learn/study) can be combined with active voice *CaCaC* template to produce *lamad* (he studied)
- *CiCeC* => *limed* (he taught)
- *CuCaC* => *lumad* (he was taught)

Finite-State Morphological Parsing

- Parsing English morphology

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	(caught +V +PAST-PART) or (catch +V +PAST)

Stems and morphological features

Finite-State Morphological Parsing

1. **Lexicon:** the list of stems and affixes, together with basic information about them (Noun stem or Verb stem, etc.)
2. **Morphotactics:** the model of morpheme ordering that explains which classes of morphemes can follow other classes of morphemes inside a word. E.g., the rule that English plural morpheme follows the noun rather than preceding it.
3. **Orthographic rules:** these **spelling rules** are used to model the changes that occur in a word, usually when two morphemes combine (e.g., the $y \rightarrow ie$ spelling rule changes *city* + *-s* to *cities*).

Finite-State Morphological Parsing

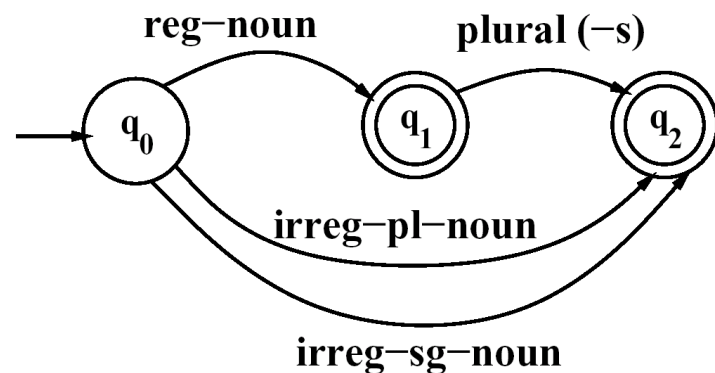
Morphological Parsing with FST

- A formal definition of FST (based on the **Mealy machine** extension to a simple FSA):
 - Q : a finite set of N states q_0, q_1, \dots, q_N
 - Σ : a finite alphabet of complex symbols. Each complex symbol is composed of an input-output pair $i : o$; one symbol i from an input alphabet I , and one symbol o from an output alphabet O , thus $\Sigma \subseteq I \times O$. I and O may each also include the epsilon symbol ϵ .
 - q_0 : the start state
 - F : the set of final states, $F \subseteq Q$
 - $\delta(q, i:o)$: the transition function or transition matrix between states. Given a state $q \in Q$ and complex symbol $i:o \in \Sigma$, $\delta(q, i:o)$ returns a new state $q' \in Q$. δ is thus a relation from $Q \times \Sigma$ to Q .

Finite-State Morphological Parsing

The Lexicon and Morphotactics

- A lexicon is a repository for words.
 - The simplest one would consist of an explicit list of every word of the language.
Inconvenient or impossible!
 - Computational lexicons are usually structured with
 - a list of each of the stems and
 - Affixes of the language together with a representation of morphotactics telling us how they can fit together.
- The most common way of modeling morphotactics is the **finite-state automaton**.

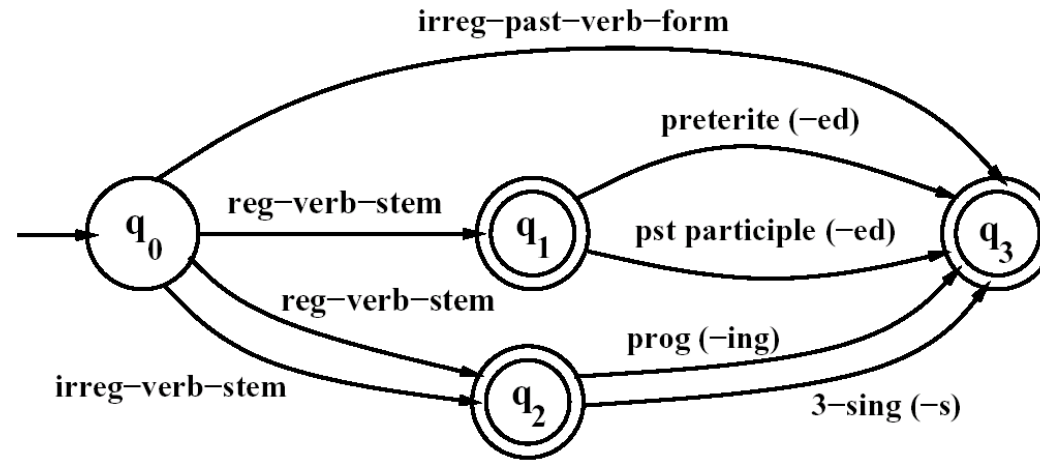


An FSA for English nominal inflection

Reg-noun	Irreg-pl-noun	Irreg-sg-noun	plural
fox	geese	goose	-s
fat	sheep	sheep	
fog	Mice	mouse	
fardvark			

Finite-State Morphological Parsing

The Lexicon and Morphotactics



Preterite: Past

Prog: present participle

An FSA for English verbal inflection

Reg-verb-stem	Irreg-verb-stem	Irreg-past-verb	past	Past-part	Pres-part	3sg
walk	cut	caught	-ed	-ed	-ing	-s
fry	speak	ate				
talk	sing	eaten				
impeach	sang					
	spoken					

The Porter Stemmer (Porter, 1980)

- A simple rule-based algorithm for stemming
- An example of a HEURISTIC method
- Based on rules like:
 - ATIONAL -> ATE (e.g., *relational* -> *relate*)
- The algorithm consists of seven sets of rules, applied in order

The Porter Stemmer: definitions

- Definitions:
 - **CONSONANT**: a letter other than A, E, I, O, U, and Y preceded by consonant (e.g. SYZYGY)
 - **VOWEL**: any other letter
- With this definition, all words are of the form:
 $(C)(VC)^m(V)$
 C=string of one or more consonants (con+)
 V=string of one or more vowels
 $m \geq 0$
- E.g.,
 - Tr ou bl e s
 - C V C V C

The Porter Stemmer: rule format

- The rules are of the form:

(condition) S1 -> S2

Where S1 and S2 are suffixes

- Conditions:

m	The measure of the stem
*S	The stem ends with S
v	The stem contains a vowel
*d	The stem ends with a double consonant
*o	The stem ends in CVC (second C not W, X, or Y)

The Porter Stemmer: Step 1

- **SSES -> SS**
 - *caresses -> caress*
- **IES -> I**
 - *ponies -> poni*
 - *ties -> ti*
- **SS -> SS**
 - *caress -> caress*
- **S -> ε**
 - *cats -> cat*

The Porter Stemmer: Step 2a (past tense, progressive)

- (m>0) EED -> EE
 - Condition verified: *agreed* -> *agree*
 - Condition not verified: *feed* -> *feed*
- (*V*) ED -> ε
 - Condition verified: *plastered* -> *plaster*
 - Condition not verified: *bled* -> *bled*
- (*V*) ING -> ε
 - Condition verified: *motoring* -> *motor*
 - Condition not verified: *sing* -> *sing*

m	The measure of the stem
*S	The stem ends with S
v	The stem contains a vowel
*d	The stem ends with a double consonant
*o	The stem ends in CVC (second C not W, X, or Y)

The Porter Stemmer: Step 2b (cleanup)

- (These rules are ran if second or third rule in 2a apply)

- **AT -> ATE**
 - *conflat(ed) -> conflate*
- **BL -> BLE**
 - *Troubl(ing) -> trouble*
- **(*d & ! (*L or *S or *Z)) -> single letter**
 - Condition verified: *hopp(ing) -> hop, tann(ed) -> tan*
 - Condition not verified: *fall(ing) -> fall*
- **(m=1 & *o) -> E**
 - Condition verified: *fil(ing) -> file*
 - Condition not verified: *fail -> fail*

Why?

m	The measure of the stem
*S	The stem ends with S
v	The stem contains a vowel
*d	The stem ends with a double consonant
*o	The stem ends in CVC (second C not W, X, or Y)

(*V*) ED -> e

Condition verified: *plastered -> plaster*

Condition not verified: *bled -> bled*

(*V*) ING -> e

Condition verified: *motoring -> motor*

Condition not verified: *sing -> sing*

The Porter Stemmer: Steps 3 and 4

- Step 3: Y Elimination (**V**) *Y* -> *I*
 - Condition verified: *happy* -> *happi*
 - Condition not verified: *sky* -> *sky*
- Step 4: Derivational Morphology, I
 - (*m*>0) *ATIONAL* -> *ATE*
 - *Relational* -> *relate*
 - (*m*>0) *IZATION* -> *IZE*
 - *generalization* -> *generalize*
 - (*m*>0) *BILITI* -> *BLE*
 - *sensibiliti* -> *sensible*

<i>m</i>	The measure of the stem
<i>*S</i>	The stem ends with <i>S</i>
<i>*v*</i>	The stem contains a vowel
<i>*d</i>	The stem ends with a double consonant
<i>*o</i>	The stem ends in <i>CVC</i> (second <i>C</i> not <i>W</i> , <i>X</i> , or <i>Y</i>)

The Porter Stemmer: Steps 5 and 6

- Step 5: Derivational Morphology, II
 - (m>0) ICATE -> IC
 - *triplicate* -> *triplic*
 - (m>0) FUL -> ϵ
 - *hopeful* -> *hope*
 - (m>0) NESS -> ϵ
 - *goodness* -> *good*
- Step 6: Derivational Morphology, III
 - (m>0) ANCE -> ϵ
 - *allowance* -> *allow*
 - (m>0) ENT -> ϵ
 - *dependent* -> *depend*
 - (m>0) ANT -> ϵ
 - *irritant* -> *irrit*
 - (m>0) IVE -> ϵ
 - *effective* -> *effect*

m	The measure of the stem
*S	The stem ends with S
v	The stem contains a vowel
*d	The stem ends with a double consonant
*o	The stem ends in CVC (second C not W, X, or Y)

The Porter Stemmer: Step 7 (cleanup)

- Step 7a
 - (m>1) E -> ϵ
 - *probate* -> *probat*
 - (m=1 & !*o) NESS -> ϵ
 - *goodness* -> *good*
- Step 7b
 - (m>1 & *d & *L) -> single letter
 - Condition verified: *control* -> *control*
 - Condition not verified: *roll* -> *roll*

m	The measure of the stem
*S	The stem ends with S
v	The stem contains a vowel
*d	The stem ends with a double consonant
*o	The stem ends in CVC (second C not W, X, or Y)

Examples

- *computers*
 - Step 1, Rule 4: -> *computer*
 - Step 6, Rule 4: -> *compute*
- *singing*
 - Step 2a, Rule 3: -> *sing*
- *controlling*
 - Step 2a, Rule 3: -> *controll*
 - Step 7b : -> *control*
- *generalizations*
 - Step 1, Rule 4: -> *generalization*
 - Step 4, Rule 11: -> *generalize*
 - Step 6, last rule: -> *general*

Problems

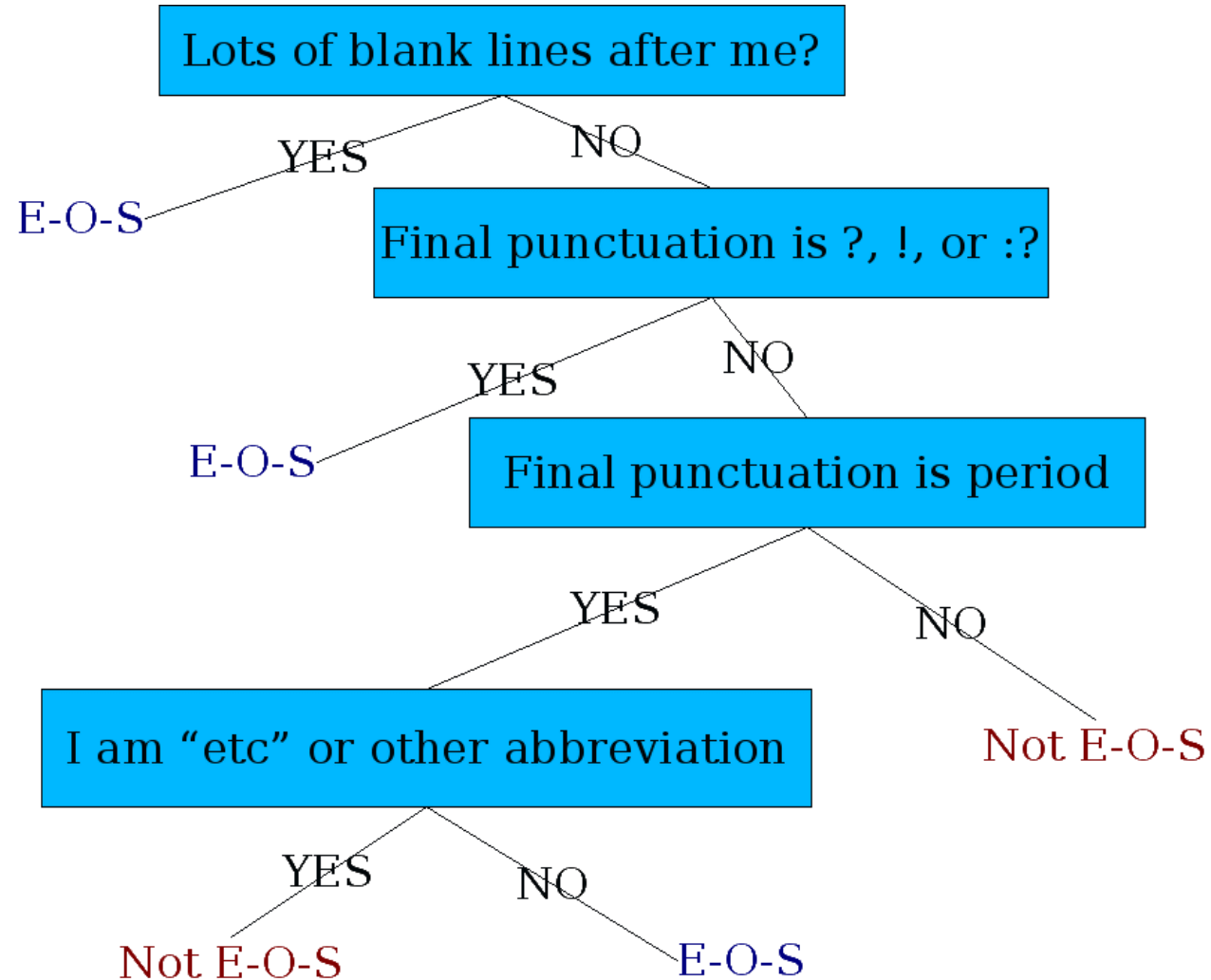
- *elephants -> eleph*
 - Step 1, Rule 4: -> *elephant*
 - Step 6, Rule 7: -> *eleph*
- *Etc.....*

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 Decision Tree



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-s)
 - Probability(word after “.” occurs at beginning-of-s)

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 Nets
 - etc.

- **Course Instructor:** Tanmoy Chakraborty (**tanmoychak.com**)
(NLP, Social Media, Graph Neural Networks)
tanchak@iitd.ac.in
- **Guest Lecture:** **Graham Neubig (CMU)**
- **Course page:** <https://sites.google.com/view/ell881-iitd/home>
- **Piazza:** <https://piazza.com/iitd.ac.in/spring2023/ell881>
- **TAs:**
 - Kshitij Alwadhi (Kshitij.Alwadhi.ee119@ee.iitd.ac.in)
 - Gurusha Juneja (ee1190480@ee.iitd.ac.in)
- **Group Email:** TBD

Spelling Error: Minimum Edit Distance



How similar are two strings?

- Spell correction

- The user typed “Appl”

Which is closest?

- App
 - Appeal
 - Apple

- Computational Biology

- Align two sequences of nucleotides

AGGCTATCACCTGACCTCCAGGCCGATGCCC
TAGCTATCACGACCGCGGGTCGATTGCCCCGAC

- Resulting alignment:

-AGGCTATCACCTGACCTCCAGGCCGA--TGCCC--
TAG-CTATCAC--GACCGC--GGTCGATTGCCCCGAC

- Also for Machine Translation, Information Extraction, Speech Recognition

Edit Distance

- The minimum edit distance between two strings
- Is the minimum number of editing operations
 - Insertion (**I**)
 - Deletion (**D**)
 - Substitution (**S**)
- Need to transform one into the other

Minimum Edit Distance

- Two strings and their **alignment**: TRIAL vs ZEIL

T	R	I	A	L
Z	E	I	*	L
s	s		d	

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
 - Distance between these is 3
- If substitutions cost 2 (Levenshtein)
 - Distance between them is 5

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

S

I

D

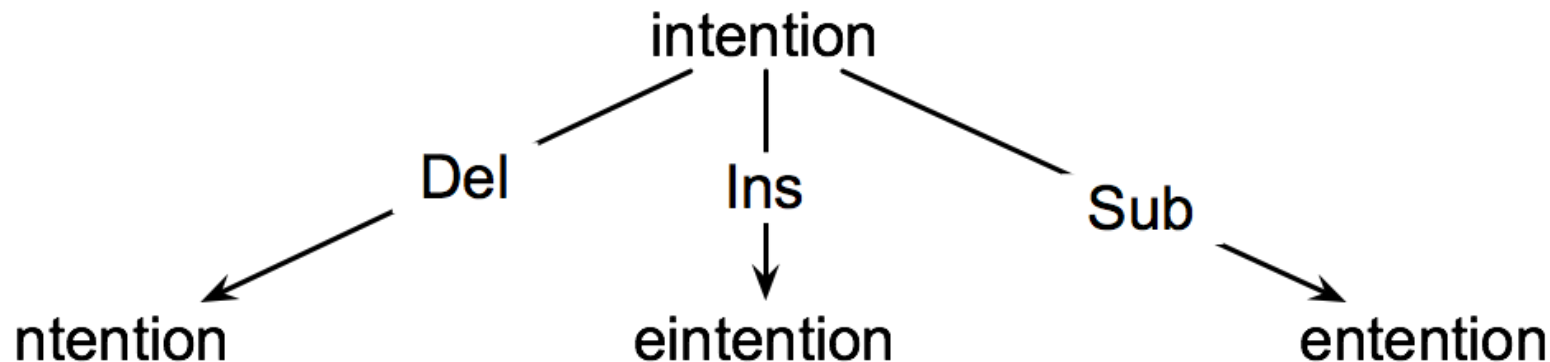
I

- **Named Entity Extraction and Entity Coreference**

- IBM Inc. announced today
- IBM profits
- US President Donald Trump announced yesterday
- for United States President Donald Trump

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

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 (Levenshtein)

- Initialization

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

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

- Recurrence Relation:

For each $i = 1 \dots M$

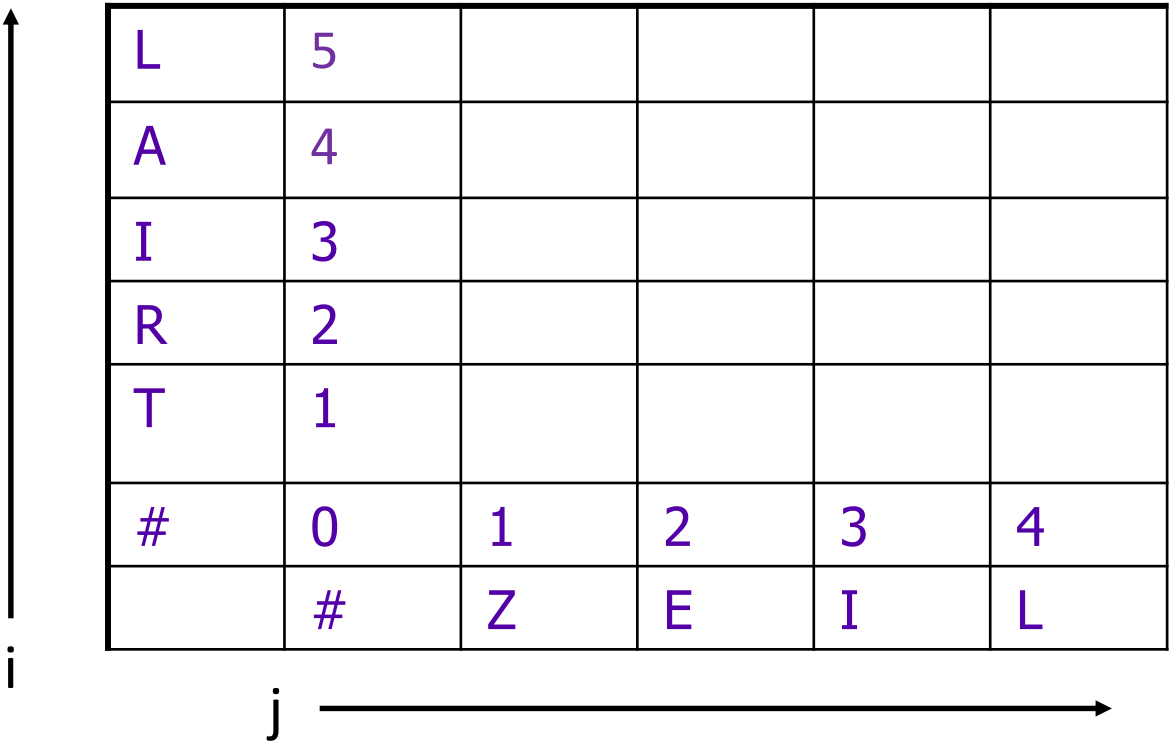
For each $j = 1 \dots N$

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

- Termination:

$D(N, M)$ is distance

The Edit Distance Table



The diagram shows an edit distance table. To the left of the table is a vertical arrow pointing upwards, labeled with the letter 'i' at its base. Below the table is a horizontal arrow pointing to the right, labeled with the letter 'j' at its start. The table itself is a 7x6 grid. The first column contains the characters 'L', 'A', 'I', 'R', 'T', '#', and an empty cell. The first row contains the values '5', '4', '3', '2', '1', '0', and '#'. The remaining cells in the first row and first column are empty. The remaining cells in the table (from row 2 to row 7 and column 2 to column 6) are all empty.

L	5				
A	4				
I	3				
R	2				
T	1				
#	0	1	2	3	4
	#	Z	E	I	L

The Edit Distance Table

<div style="display: flex; align-items: center;"> <div style="width: 10px; height: 100px; border-left: 1px solid black; margin-right: 5px;"></div> <div style="writing-mode: vertical-rl; transform: rotate(180deg);">i</div> </div>	5	L	5	6	7	6	5
	4	A	4	5	6	5	6
	3	I	3	4	5	4	5
	2	R	2	3	4	5	6
	1	T	1	2	3	4	5
	0	#	0	1	2	3	4
			#	Z	E	I	L
			0	1	2	3	4
			j				

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

The Edit Distance Table

(INTENSION vs EXECUTION)

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
- 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

Edit Distance with Backtrace

T R I A L
| | | | |
Z E I * L
s s d





L	5	6	7	6	5
A	4	5	6	5	6
I	3	4	5	4	5
R	2	3	4	5	6
T	1	2	3	4	5
#	0	1	2	3	4
	#	Z	E	I	L

- ← Substitute
- ← No Change
- ← Delete

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

Edit Distance with Backtrace (Another Path)

L	5	6	7	6	5
A	4	5	6	5	6
I	3	4	5	4	5
R	2	3	4	5	6
T	1	2	3	4	5
#	0	1	2	3	4
	#	Z	E	I	L

-  Substitute
-  No Change
-  Delete
-  Insert

Cost is same, i.e., 5

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

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:

$D(N, M)$ is distance

- Recurrence Relation:

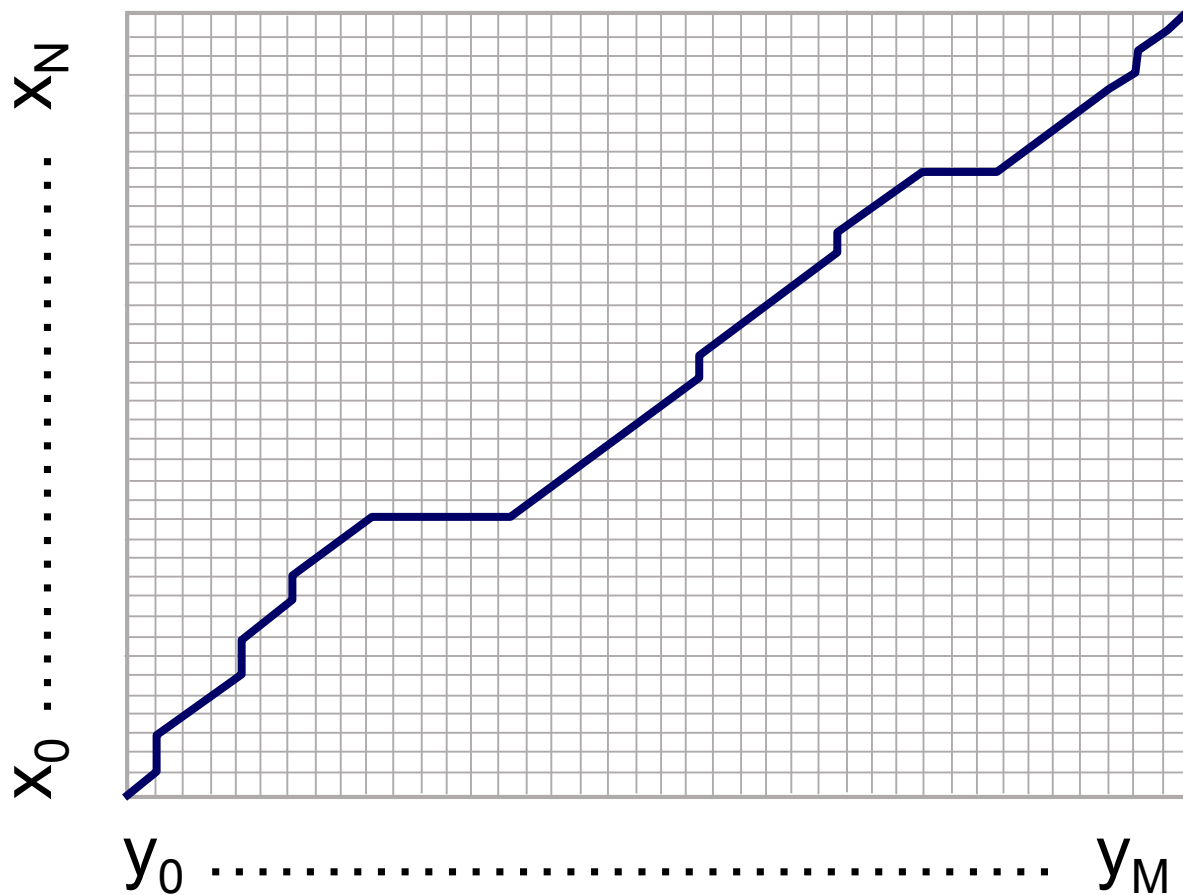
For each $i = 1 \dots M$

For each $j = 1 \dots N$

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

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

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

Performance

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