

Faculty for Informatics

Technical
University
of Munich



Natural Language Processing

IN2361

PD Dr. Georg Groh

Social Computing
Research Group

Chapter 2

Regular Expressions, Text-Normalization, Edit-Distance

- content is based on [1] and [2]
- certain elements (e.g. equations or tables) were taken over or taken over in a modified form from [1] or [2]
- citations of [1] and [2] or from [1] or [2] are omitted for legibility
- errors are fully in the responsibility of Georg Groh
- BIG thanks to Dan and James for a great book!

Regular Expressions

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.

ELIZA Conversation 1966

```
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/
```

Regular Expressions

- **Regular Expression**: algebraic notation for characterizing a set of strings
→ simple deterministic classifier / means for search

[]: **disjunction** of letters:

Pattern	Matches
[wW]oodchuck	Woodchuck <i>or</i> woodchuck
[1234567890]	<i>a single digit</i>

[-]: **ranges**:

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

Regular Expressions

caret: [^]: **negation** (if first in []):

Pattern	Matches	
[^A-Z]	<i>not an upper case letter</i>	O <u>y</u> fn pripetchik
[^Ss]	<i>neither 'S' nor 's'</i>	<u>I</u> have no exquisite reason
[e^]	<i>either e or ^</i>	<u>^</u> ^ee Look here
[^e^]	<i>neither e nor ^</i>	^^ee <u>L</u> ook here
a^b	<i>the pattern a caret b</i>	Look up <u>a^b</u> now

|: **disjunction**:

Pattern	Matches
yours mine	yours or mine
a b c	same as [abc]
[gG]roundhog [Ww]oodchuck	obvious
gupp(y ies)	guppy or guppies

Regular Expressions

? * + { } **quantifiers** (for counting); . wildcards

Pattern	Matches	
colou?r	<i>optional previous char or expr</i>	<u>color</u> or <u>colour</u>
o*h!	<i>zero or more of previous char or expr</i>	<u>h!</u> or <u>oh!</u> or <u>ooh!</u> ...
o+h!	<i>one or more of previous char or expr</i>	<u>oh!</u> or <u>ooh!</u> or <u>oooh!</u> ...
a{3,5}	<i>{x,y} : exactly x to y many</i>	<u>aaa</u> or <u>aaaa</u> or <u>aaaaa</u>
beg.n	<i>. matches any char except \r</i>	<u>begin</u> or <u>begun</u> or <u>beg3n</u> ...

^ \$ \b \B **anchors**:

Pattern	Matches	
^[A-Z]	<i>at start of a line</i>	<u>P</u> alo Alto
^[^A-Za-z]		<u>1</u> "Hello"
\.\$	<i>at end of the line</i>	The end <u>.</u>
.\$		The end <u>?</u> The end <u>!</u>
\bthe\b	<i>matches word boundaries</i>	<u>the</u> world <i>but not</i> <u>other</u>

any sequence of digits,
underscores, or letters

Basic (application specific !) tasks:

- Segmenting / tokenizing words in running text
- Normalizing word formats (e.g. lemmatization)
- Segmenting sentences in running text

Example: How Many Words?

I do uh main- mainly business data processing

- **disfluencies** in utterances: **fragments**, fillers, (similar also: **emoticons**) etc.

*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

Example: How Many Words?

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

- **Type**: an **element** of the **vocabulary** V .
- **Token**: an **instance** of that type in running text. Number of Tokens: **N**
- How many in example? 15 tokens , 13 types

Corpora:

Corpus	Tokens = N	Types = $ V $
Shakespeare	884 thousand	31 thousand
Brown corpus	1 million	38 thousand
Switchboard telephone conversations	2.4 million	20 thousand
COCA	440 million	2 million
Google N-grams	1 trillion	13 million

Heran's / Heap's Law:

$$|V| = kN^{\beta}$$

$$0 < \beta < 1 \quad , \quad k > 0$$

$$\beta \approx 0.7$$

Word Tokenization

- **Tokenization**: segment text into words

- **Issues:**

- *Finland's capital* → Finland Finlands Finland's ?
- *what're, I'm, isn't* → what are, I am, is not
- *\$ 4.99* → \$4.99 \$ 4 99 ?
- *state-of-the-art* → state of the art ?
- *lowercase* → lower-case lowercase lower case ?
- *San Francisco* → one token or two? (\leftrightarrow NER)
- *m.p.h., Ph.D., AT&T* → keep together?
- *www.google.de* → http://www.google.de ?
- *233,455* → 233.455 ? 233455 ?
- count punctuation as separate words?
- wtf, lol, ☺, :-) →

- **Penn Treebank tokenization 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	”	.

Tokenization – Language Specific Issues

- French

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

- German

- noun compounds are not segmented:
example: *Lebensversicherungsgesellschaftsangestellter*
(‘life insurance company employee’)
 - → German information retrieval needs **compound splitter**
-

- 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

- treating all **issues of tokenization** by rules + automata (fast) is complicated → use **ML**-based sequence processing.
but is there a third, more simple, data-driven way?
- → **Byte Pair Encoding (BPE)**:
 - **iteratively merge frequent pairs of characters**:
 - start with symbol-vocabulary of characters + end-of-word-character.
 - for most frequent character n-gram pair in words: create new n-gram.
 - iterate. → word segmentation into character n-grams
- **benefits**: example: *low*, *lowest* in training set but *lower* not in training set but in test set → *lower* is decomposed as *low* + *er*: *low* and *er* may be known → **compositional representation of unknowns**

- iteratively merge frequent pairs of characters:
 - start with symbol-vocabulary of characters + end-of-word-character.
 - for most frequent character n-gram pair in words: create new n-gram.
 - iterate. → word segmentation into character n-grams

#occ	word
5	low_
2	lowest_
6	newer_
3	wider_
2	new_

_ , d, e, i, l, n, o, r, s, t, w

all characters in vocab

_ : end of word token

- iteratively merge frequent pairs of characters:
 - start with symbol-vocabulary of characters + end-of-word-character.
 - for most frequent character n-gram pair in words: create new n-gram.
 - iterate. → word segmentation into character n-grams

#occ	word
5	low_
2	lowest_
6	newer_
3	wider_
2	new_

_ , d, e, i, l, n, o, r, s, t, w, r_

most frequent pair: r_ (#occ=9)

- iteratively merge frequent pairs of characters:
 - start with symbol-vocabulary of characters + end-of-word-character.
 - for most frequent character n-gram pair in words: create new n-gram.
 - iterate. → word segmentation into character n-grams

#occ	word
5	low_
2	lowest_
6	newer_
3	wider_
2	new_

_ , d, e, i, l, n, o, r, s, t, w, r_ , er_

most frequent pair: e r_ (#occ=9)

- iteratively merge frequent pairs of characters:
 - start with symbol-vocabulary of characters + end-of-word-character.
 - for most frequent character n-gram pair in words: create new n-gram.
 - iterate. → word segmentation into character n-grams

#occ	word
5	l o w _
2	l o w e s t _
6	n ew er _
3	w i d er _
2	n ew _

_ , d, e, i, l, n, o, r, s, t, w, **r_** , **er_** , **ew**

most frequent pair: **ew** (#occ=8)

Byte Pair Encoding

next merge	current “vocabulary”
(n, ew)	_ , d, e, i, l, n, o, r, s, t, w, r_ , er_ , ew, new
(l, o)	_ , d, e, i, l, n, o, r, s, t, w, r_ , er_ , ew, new, lo
(lo, w)	_ , d, e, i, l, n, o, r, s, t, w, r_ , er_ , ew, new, lo, low
(new, er_)	_ , d, e, i, l, n, o, r, s, t, w, r_ , er_ , ew, new, lo, low, newer_
(low, _)	_ , d, e, i, l, n, o, r, s, t, w, r_ , er_ , ew, new, lo, low, newer_ , low_

l o w e r _ \rightarrow low er_ (via greedy longest match first decoding (maximum matching, MaxMatch)) after learning token set

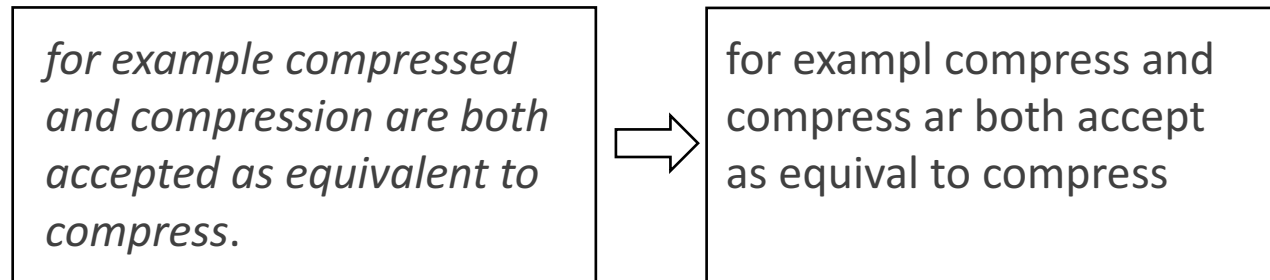
- algorithm’s meta **parameter: number k of merge steps**: if k large \rightarrow most words and affixes will get their own character N-gram representation
- alternative: **wordpiece algorithm** (e.g. used for BERT): merge pairs not based on frequency but in terms of **maximizing the likelihood of the resulting language model of word pieces** (while minimizing the number of wordpieces)

- **Normalization**: mapping words / tokens in a standard format
→ create equivalence classes: {U.S. , U.S.A, USA} → USA
- possible element: **Case-Folding / Lowercasing**:
not always helpful :
 - *GOOD LORD! it's US, you fool! not you alone!*
(<-> sentiment analysis, information extraction,)
 - The US government announced .. vs. it was us that had the trouble
(<-> information extraction, information retrieval...)
- **standard algorithms for tokenization + normalization**:
 - deterministic algorithms based on **regular expressions**
compiled into very efficient finite state automata.
 - also possible: **ML-based** algos trained on large hand segmented corpora

- 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*
 - most sophisticated classic method: **morphological parsing**
 - **Morphology**: study of way words are built up from smaller meaning-bearing units called
 - **Morphemes**:
 - **Stems**: The core meaning-bearing units
 - **Affixes**: Bits and pieces that adhere to stems
- example: *cats* → stem: *cat*, affix: *s*

- “Poor man’s lemmatization” : reduce terms to their stems (mostly in / from information retrieval)
- **Stemming**: crude chopping of affixes (language dependent)

Example:



Stemming for English: Porter's Stemmer

Set of **term rewriting rules**, applied to words repeatedly in passes

Step 1a

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

Step 2 (for long stems)

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

Step 1b

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

Step 3 (for longer stems)

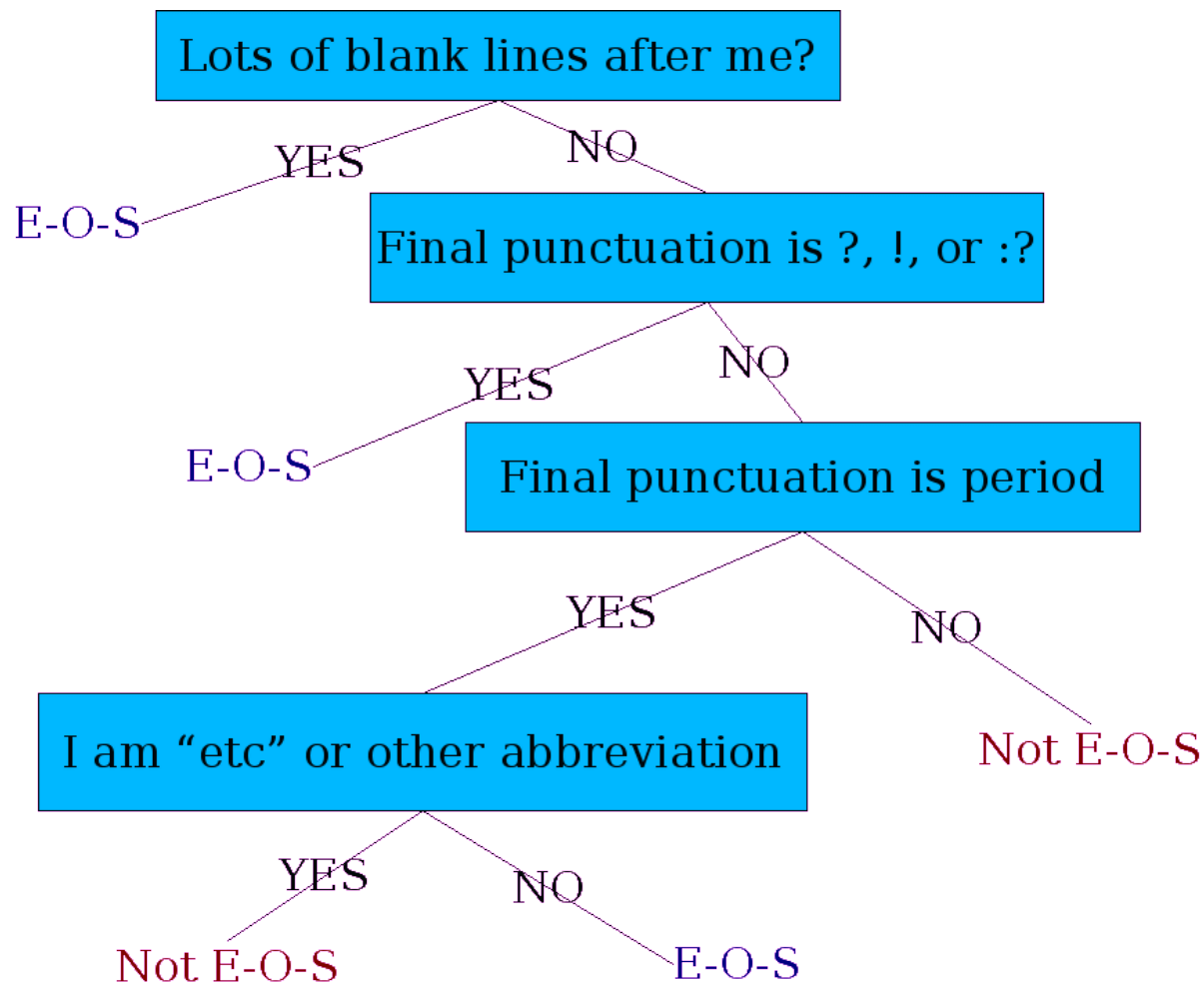
al	→ ∅	revival	→ reviv
able	→ ∅	adjustable	→ adjust
ate	→ ∅	activate	→ activ
...			

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

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**
 - For each “.” : apply classifier:
features from “.”’s neighborhood →
class1: endOfSentence or
class2: NotEndOfSentence
 - Classifiers: hand-written rules, regular expressions, or machine-learning

Sentence Segmentation with Decision Tree



Edit Distance and Alignment

How similar are two strings?

- Spell correction

- the user typed “*graffe*”:
which word is closest?

- *graf*
- *graft*
- *grail*
- *giraffe*

- Computational Biology

- Align two sequences of nucleotides

```
AGGCTATCACCTGACCTCCAGGCCGATGCCC  
TAGCTATCACGACCGCGGTCGATTTGCCCGAC
```

- Resulting alignment:

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

- Also for Machine Translation, Information Extraction, Speech Recognition

Edit Distance

- **Minimum Edit Distance** between two strings is the minimum number of editing operations
 - insertion
 - deletion
 - substitutionneeded to transform one into the other

example:

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

} optimal alignment
btw. intention
and execution

- **Levenshtein distance:**
assign costs:
 - if each operation has cost of 1 \rightarrow dist = 5
 - if substitutions cost 2 \rightarrow dist = 8

Algorithm for MinEditDistance

- **Todo:** search for **shortest weighted path**
(sequence of edits with minimal overall cost)
from start string **X** (length **n**) to target string **Y** (length **m**)
- Define: $D[i,j]$ = minEditDist btw.
first i characters of **X** ($X[1..i]$) and
first j characters of **Y** ($Y[1..j]$)
- $\rightarrow \text{minEditDist}(X, Y) = D[m,n]$
- Approach: **Dynamic Programming:**
compute $D(i,j)$ for small i,j and compute larger $D(i,j)$ based on previously
computed smaller values

Algorithm for MinEditDistance

function MIN-EDIT-DISTANCE(*source*, *target*) **returns** *min-distance*

$n \leftarrow \text{LENGTH}(\textit{source})$

$m \leftarrow \text{LENGTH}(\textit{target})$

Create a distance matrix $\textit{distance}[n+1, m+1]$

Initialization: the zeroth row and column is the distance from the empty string

$D[0,0] = 0$

for each row i **from** 1 **to** n **do**

$D[i,0] \leftarrow D[i-1,0] + \textit{del-cost}(\textit{source}[i])$

for each column j **from** 1 **to** m **do**

$D[0,j] \leftarrow D[0,j-1] + \textit{ins-cost}(\textit{target}[j])$

Recurrence relation:

for each row i **from** 1 **to** n **do**

for each column j **from** 1 **to** m **do**

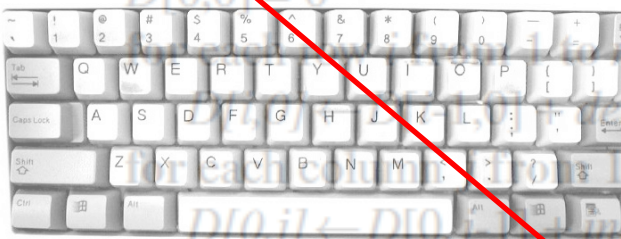
$D[i,j] \leftarrow \text{MIN}(D[i-1,j] + \textit{del-cost}(\textit{source}[i]),$
 $D[i-1,j-1] + \textit{sub-cost}(\textit{source}[i], \textit{target}[j]),$
 $D[i,j-1] + \textit{ins-cost}(\textit{target}[j]))$

Termination

return $D[n,m]$

Algorithm for MinEditDistance

option: may want to choose non-uniform costs here



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

Recurrence relation:
for each row i from 1 to n do
for each column j from 1 to m do

$$D[i,j] \leftarrow \text{MIN}(D[i-1,j] + \text{del-cost}(\text{source}[i]), \\ D[i-1,j-1] + \text{sub-cost}(\text{source}[i], \text{target}[j]), \\ D[i,j-1] + \text{ins-cost}(\text{target}[j]))$$

Termination
return D[n,m]

Algorithm for MinEditDistance (Levenshtein)

- Initialization

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

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

- Recurrence Relation:

For each $i = 1 \dots M$

For each $j = 1 \dots N$

$$D(i, j) = \min \begin{cases} D(i-1, j) + 1 & \text{ } i \rightarrow i-1 : \text{a deletion from source is necessary} \\ D(i, j-1) + 1 & \text{ } j-1 \rightarrow j : \text{an insertion into target is necessary} \\ D(i-1, j-1) + \begin{cases} 2; & \text{if } X(i) \neq Y(j) \\ 0; & \text{if } X(i) = Y(j) \end{cases} \end{cases}$$

- Termination:

$D(N, M)$ is distance

Algorithm for MinEditDistance (Levenshtein)

$j \Rightarrow$

$i \Downarrow$

Src\Tar	#	e	x	e	c	u	t	i	o	n
#	0	1	2	3	4	5	6	7	8	9
i	1	2	3	4	5	6	7	6	7	8
n	2	3	4	5	6	7	8	7	8	7
t	3	4	5	6	7	8	7	8	9	8
e	4	3	4	5	6	7	8	9	10	9
n	5	4	5	6	7	8	9	10	11	10
t	6	5	6	7	8	9	8	9	10	11
i	7	6	7	8	9	10	9	8	9	10
o	8	7	8	9	10	11	10	9	8	9
n	9	8	9	10	11	12	11	10	9	8

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

Extension of Algorithm: Compute Alignment

- Alignment of X and Y: from (n,m): reconstruct **non-decreasing path through matrix** (“backtrace”): **remember** in each step “**where we came from**”

tar j \Rightarrow

src

i
↓

	#	e	x	e	c	u	t	i	o	n
#	0	1	2	3	4	5	6	7	8	9
i	1	$\nwarrow \leftarrow \uparrow 2$	$\nwarrow \leftarrow \uparrow 3$	$\nwarrow \leftarrow \uparrow 4$	$\nwarrow \leftarrow \uparrow 5$	$\nwarrow \leftarrow \uparrow 6$	$\nwarrow \leftarrow \uparrow 7$	$\nwarrow 6$	$\leftarrow 7$	$\leftarrow 8$
n	2	$\nwarrow \leftarrow \uparrow 3$	$\nwarrow \leftarrow \uparrow 4$	$\nwarrow \leftarrow \uparrow 5$	$\nwarrow \leftarrow \uparrow 6$	$\nwarrow \leftarrow \uparrow 7$	$\nwarrow \leftarrow \uparrow 8$	$\uparrow 7$	$\nwarrow \leftarrow \uparrow 8$	$\nwarrow 7$
t	3	$\nwarrow \leftarrow \uparrow 4$	$\nwarrow \leftarrow \uparrow 5$	$\nwarrow \leftarrow \uparrow 6$	$\nwarrow \leftarrow \uparrow 7$	$\nwarrow \leftarrow \uparrow 8$	$\nwarrow 7$	$\leftarrow \uparrow 8$	$\nwarrow \leftarrow \uparrow 9$	$\uparrow 8$
e	4	$\nwarrow 3$	$\leftarrow 4$	$\nwarrow \leftarrow 5$	$\leftarrow 6$	$\leftarrow 7$	$\leftarrow \uparrow 8$	$\nwarrow \leftarrow \uparrow 9$	$\nwarrow \leftarrow \uparrow 10$	$\uparrow 9$
n	5	$\uparrow 4$	$\nwarrow \leftarrow \uparrow 5$	$\nwarrow \leftarrow \uparrow 6$	$\nwarrow \leftarrow \uparrow 7$	$\nwarrow \leftarrow \uparrow 8$	$\nwarrow \leftarrow \uparrow 9$	$\nwarrow \leftarrow \uparrow 10$	$\nwarrow \leftarrow \uparrow 11$	$\nwarrow \uparrow 10$
t	6	$\uparrow 5$	$\nwarrow \leftarrow \uparrow 6$	$\nwarrow \leftarrow \uparrow 7$	$\nwarrow \leftarrow \uparrow 8$	$\nwarrow \leftarrow \uparrow 9$	$\nwarrow 8$	$\leftarrow 9$	$\leftarrow 10$	$\leftarrow \uparrow 11$
i	7	$\uparrow 6$	$\nwarrow \leftarrow \uparrow 7$	$\nwarrow \leftarrow \uparrow 8$	$\nwarrow \leftarrow \uparrow 9$	$\nwarrow \leftarrow \uparrow 10$	$\uparrow 9$	$\nwarrow 8$	$\leftarrow 9$	$\leftarrow 10$
o	8	$\uparrow 7$	$\nwarrow \leftarrow \uparrow 8$	$\nwarrow \leftarrow \uparrow 9$	$\nwarrow \leftarrow \uparrow 10$	$\nwarrow \leftarrow \uparrow 11$	$\uparrow 10$	$\uparrow 9$	$\nwarrow 8$	$\leftarrow 9$
n	9	$\uparrow 8$	$\nwarrow \leftarrow \uparrow 9$	$\nwarrow \leftarrow \uparrow 10$	$\nwarrow \leftarrow \uparrow 11$	$\nwarrow \leftarrow \uparrow 12$	$\uparrow 11$	$\uparrow 10$	$\uparrow 9$	$\nwarrow 8$

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

$$\text{ptr}(i, j) = \begin{cases} \text{to } \text{UP} & \text{deletion} \\ \text{from } \text{LEFT} & \text{insertion} \\ \text{from } \text{DIAG} & \text{substitution or id} \end{cases}$$



- (1) Dan Jurafsky and James Martin: Speech and Language Processing (3rd ed. draft, version October, 2019); Online: <https://web.stanford.edu/~jurafsky/slp3/> (URL, Oct 2019) (this slideset is especially based on chapter 2)
- (2) Powerpoint slides from Dan Jurafsky and James Martin: Speech and Language Processing (3rd ed. draft); Online: <https://web.stanford.edu/~jurafsky/slp3/> (URL, Oct 2018)

Recommendations for Studying

- minimal approach:

work with the slides and understand their contents! Think beyond instead of merely memorizing the contents

- standard approach:

minimal approach + read the corresponding pages in Jurafsky [1]

- interested students

== standard approach