CIT
Social Computing
Research Group

Technical
University
of Munich

Natural Language Processing
IN2361

Prof. Dr. Georg Groh

Organisation:

- Wed + Fr 14-16, Interim 1 HS 2, all recorded on RBG Live
- Moodle with all materials.
- complete literature: freely available on Web. Every lecture has corresponding background reading
- no tutorials
- 120 min live written exam + repeat exam

- further elements from our NLP universe:
  - NLP 2.0 (Advanced Deep Learning: NLP) in SS
  - Lab course NLP, 10 ECTS, in SS: 50-60 seats
  - Lab course Ethical AI, 10 ECTS, in SS: 20 seats
  - Lab course Explainable AI, 10 ECTS, in WS: 20-40 seats
  - Seminar Ethical AI, 5 ECTS, in SS: 20 seats.

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_1:	Men are all alike.
ELIZA_1:	IN WHAT WAY
User_2:	They're always bugging us about something or other.
ELIZA_2:	CAN YOU THINK OF A SPECIFIC EXAMPLE
User_3:	Well, my boyfriend made me come here.
ELIZA_3:	YOUR BOYFRIEND MADE YOU COME HERE
User_4:	He says I'm depressed much of the time.
ELIZA_4:	I AM SORRY TO HEAR YOU ARE DEPRESSED

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/
```

s/xxx/yyy/ replaces xxx with yyy

Regular Expressions

- Regular Expression: algebraic notation for characterizing a set of strings
  → simple deterministic classifier (e.g. for searching) or for substitutions

[ ]: disjunction of letters:

Pattern	Matches
`[wW]oodchuck`	`Woodchuck` *or* `woodchuck`
`[1234567890]`	*a single digit*

[-]: ranges:

Pattern	Matches	
`[A-Z]`	*an upper case letter*	`Drenched Blossoms`
`[a-z]`	*a lower case letter*	`my beans were impatient`
`[0-9]`	*a single digit*	`Chapter 1: Down the Rabbit Hole`

In latest edition of Jurafsky, RegEx are enclosed in /… / (e.g. /`[wW]oodchuck`/

Regular Expressions

caret: [^]: negation (if first in [ ]): you need to put first otherwise it is a character

Pattern	Matches	
[^A-Z]	*not an upper case letter*	Oyfn pripetchik
[^Ss]	*neither 'S' nor 's'*	I have no exquisite reason
[e^]	*either e or ^*	^^ee Look here
[^e^]	*neither e nor ^*	^^ee Look here
a^b	*the pattern a caret b*	Look up a^b 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

star is zero or more char

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

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

^ $ \b \B anchors:

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

any sequence of digits, underscores, or letters

Text Normalization

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 / Heaps' Law:

Which words should be in vocabulary formula

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

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

$$\beta \approx 0.7$$

Word Tokenization

- **Tokenization**: segment text into words
- Issues:
    - *Finland's capital* → Finland Finlands Finland 's ?
    - clitics: *what're, I'm, isn't* → what are, I am, is not, what 're, is n't ?
    - *$ 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? (←→ 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
    - "character-based" "radical based"…..?

Byte Pair Encoding

- treating all issues of tokenization by rules + automata (fast) is complicated → use ML-based (neural) sequence models with hand-labelled training data.
  But is there a third, more simple, data-driven way, bridging btw. word-level and character-level?

- → 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

Byte Pair Encoding

- 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

En cok tekrar eden karakter ikililerine bakiyor
ilk basta r_ mesela 9 kere r_ yeni bir karakterdir diyor

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

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

all characters in vocab

_ : end of word token

Byte Pair Encoding

- 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 e w e r_
3	w i d e r_
2	n e w _

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

most frequent pair: r _ (#occ=9)

Byte Pair Encoding

- 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 e w er_
3	w i d er_
2	n e w _

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

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

Byte Pair Encoding

- 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 _ → 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 → 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)

Word Normalization

- **Normalization**: mapping words / tokens in a standard format
  → create equivalence classes: {U.S. , U.S.A, USA} $\rightarrow$ 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 models trained on large hand segmented
    corpora

Lemmatization

- Reduce inflections or variant forms to base form
  - *am, are, is* $\rightarrow$ *be*
  - *car, cars, car's, cars'* $\rightarrow$ *car*
  - *the boy's cars are different colors* $\rightarrow$ *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 $\rightarrow$ stem: *cat,* affix: *s*

Stemming

- "Poor man's lemmatization" : reduce terms to their stems (mostly in / from information retrieval)

- Stemming: crude chopping of affixes (language dependent)

  Example:

for example compressed and compression are both accepted as equivalent to compress.	⇨	for exampl compress and compress ar both accept as equival to compress

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    → ø     cats     → cat
```

Step 2 (for long stems)

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

Step 1b

```
(*v*)ing → ø  walking   → walk
              sing      → sing
(*v*)ed  → ø  plastered → plaster
…
```

Step 3 (for longer stems)

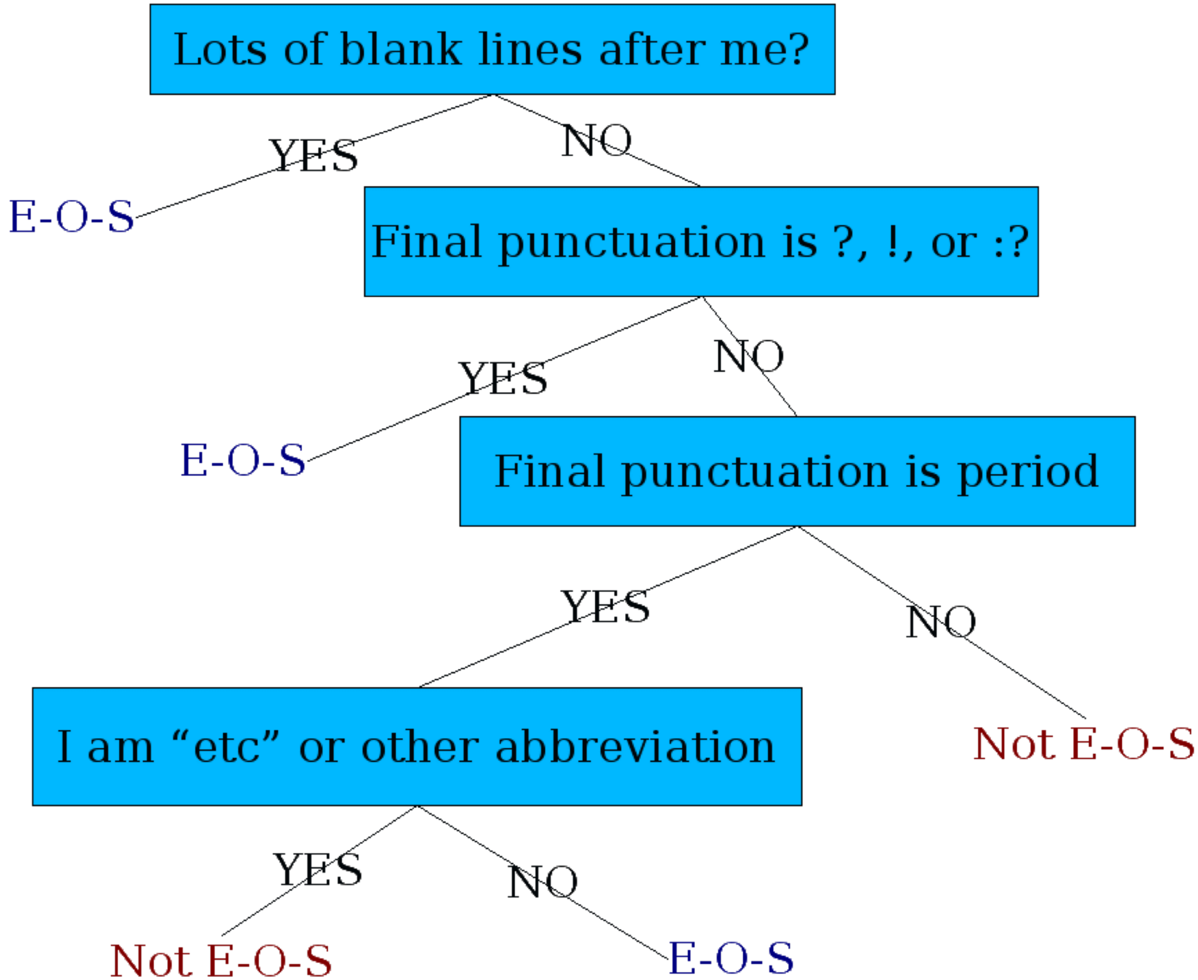
```
al   → ø  revival    → reviv
able → ø  adjustable → adjust
ate  → ø  activate   → activ
…
```

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
  - substitution

  needed 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:** if each operation has cost of 1 → dist = 5
  assign costs: if substitutions cost 2 → 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])

- → 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*

$n \leftarrow$ LENGTH(*source*)
$m \leftarrow$ LENGTH(*target*)
Create a distance matrix *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**
$D[i,0] \leftarrow D[i\text{-}1,0] + \textit{del-cost}(\textit{source}[i])$
for each column j **from** 1 **to** m **do**
$D[0,j] \leftarrow D[0,j\text{-}1] + \textit{ins-cost}(\textit{target}[j])$

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

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

\# *Termination*
return D[n,m]

Algorithm for MinEditDistance

function MIN-EDIT-DISTANCE(*sour*

option: may want to choose non-uniform costs here

Create a distance matrix *distance[n*

\# *Initialization: the zeroth row and*

$D[0,0] = 0$

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

$D[i,0] \leftarrow D[i-1,0] + \textit{del-cos}$

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

$D[0,j] \leftarrow D[0,j-1] + \textit{ins-cos}$

Confusion matrix for spelling errors

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

Y (correct)

\# *Recurrence relation:*

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

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

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

\# *Termination*

return $D[n,m]$

Algorithm for MinEditDistance (Levenshtein)

- Initialization

  D(i,0) = i

  D(0,j) = j

- Recurrence Relation*:*

  For each i = 1…M

  For each j = 1…N

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

i-1 → i : another deletion from source is necessary

j-1 → j : another insertion into target is necessary

- Termination*:*

  D(N,M) is distance

Algorithm for MinEditDistance (Levenshtein)

j

i

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

first column always like that

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

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 ⇨

src

i ⇩

	#	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**

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

$$ptr(i,j) = \begin{cases} \uparrow \text{ UP} & \text{deletion was min} \\ \leftarrow \text{ LEFT} & \text{insertion was min} \\ \nwarrow \text{ DIAG} & \text{substitution or id was min} \end{cases}$$

Bibliography

(1) Dan Jurafsky and James Martin: Speech and Language Processing (3rd ed. draft, version Jan, 2022); Online: https://web.stanford.edu/~jurafsky/slp3/ (URL, Oct 2022) (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 2022)

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 + do a selection of the exercises in Jurafsky [1]