

IN THE NAME OF GOD

PRE-PROCESSING

What is word

- Tokenization

- Process of splitting an input sequence into so-called tokens.
- Tokens: word, sentence, paragraph,..(what are the token boundries of each if these?)
- How to find boundries of words:

- In english: space or punctuation
- In german there are compound words which are written without spaces
 - Rechtsschutzversicherungsgesellschaften -> insure companies which provide legal protection

- In Japanes there are no spaces at all

- در انگلیسی یا فارسی، گاهی فاصله بیانگر مرز کلمه نیست. مثل مکتب خانه. یا hot dog . در فارسی، عدم رعایت نیم فاصله، منجر به شکلگیری کلمات اشتباه میشود.
- الزاما punctuation نشاندهنده مرز نیست. مانند نقطه یا ممیز(در زبان فارسی) برای بیان عدد اعشاری... تاریخ...

Nltk.tokenize

- This is Andrew's text, isn't it?
- `Nltk.tokenize.whitespaceTokenizer`:
 - this-is-andrew's-text,-isn't-it?
 - It and it? Two different tokens with the same meaning
- `Nltk.tokenize.wordPunctTokenizer`
 - this-is-andrew-'-s-text-,-isn-'-t-it-?
 - Isn, t aren't very meaningfull
- `Nltk.tokenize.TreebankWordTokenizer`
 - this-is-andrew-'s-text-,-is-n't-it-?

hazm

- `From hazm import *`
- `sent_tokenize('ما هم برای وصل کردن آمدیم! ولی برای پردازش، جدا جدا بهتر نیست؟')`
 - `['ولی برای پردازش، جدا بهتر نیست؟', 'اما هم برای وصل کردن آمدیم']`
- `word_tokenize('ولی برای پردازش، جدا بهتر نیست؟')`
 - `['?', 'نیست', 'بهتر', 'جدا', ',', 'پردازش', 'برای', 'ولی']`

Token normalization

- Get same token for different forms of word
 - Wolf, wolves - talk, talks - forget, forgotten
- Stemming
 - Hueristics for removing and replacing suffixes to get to the root form (Stem) of word. computation-> comput, feet->feet
 - [Nltk.stem.PorterStemmet](#)
- Lemmatization
 - Doing normalization properly using vocabulary and morphological analysis. Returns dictionary form (lemma) of word. Computation->compute, feet-> foot
 - [Nltk.stem.WordNetLemmatizer](#)

- Normalize capital letters

- Us, us Us, US ?

- (heuristic: lowercasing the beginning of sentence, and words of title. Leave mid-sentence words as they are)

- Acronyms

- eta, e.t.a, E.T.A (using regular expressions)

- در زبان فارسی کاراکترهای، ی ، ک گاهی با کاراکترهای متفاوتی در صفحات وب نوشته میشود.
 - مسئله همزه. (مسئله، مسأله.) نیم فاصله، کسره اضافه در کلمه ختم به ه. کلمه ی ختم به ه.

- `normalizer = hazm.Normalizer()`
- `normalizer.normalize('اصلاح نویسه ها و استفاده از نیمفاصله پردازش را آسان می کند')`
- 'اصلاح نویسه ها و استفاده از نیمفاصله پردازش را آسان می کند'
- `stemmer = hazm.Stemmer()`
- `stemmer.stem('کتابها')`
- 'کتاب'
- `lemmatizer = hazm.Lemmatizer()`
- `lemmatizer.lemmatize('می روم')`
- 'رفت#رو'

2-SPELL CHECKING

Spelling Correction and Edit Distance

- Non-word error detection:
 - detecting “graffe” “سوزن”, “مصواک”, “مدا”
- Non-word error correction:
 - figuring out that “graffe” should be “giraffe”
- Context-dependent error detection and correction:
 - Figuring out that “war and piece” should be peace
 - “حرکات ارضی و طولی”, “غصه هایی پر از غم و شادی”

Non-Word Error Detection

- Any word not in a dictionary is a spelling error
- Need a big dictionary!
 - The tradeoff (for rare words – wont, veery)
- What to use?
 - FST dictionary!!

Isolated Word Error Correction

- How do I fix “**graffe**”?
 - Search through all words:
 - graf
 - craft
 - grail
 - giraffe
 - Pick the one that's closest to **graffe**
 - What does “closest” mean?
 - We need a **distance metric**.
 - The simplest one: **edit distance**

Edit Distance

- The minimum edit distance between two strings is the minimum number of editing operations Needed to transform one string into the other
- operations
 - Insertion
 - Deletion
 - Substitution

An example

- Andrew
- Andrewz
 - 1. substitute m to n
 - 2. delete the z
- Distance = 3

String distance metrics: *Levenshtein*

- Given strings s and t
 - Distance is **shortest sequence of edit commands that transform s to t**
 - Simple set of operations:
 - Copy character from s over to t (*cost 0*)
 - Delete a character in s (*cost 1*)
 - Insert a character in t (*cost 1*)
 - Substitute one character for another (*cost 2*)
- This is “Levenshtein distance”

نکته: در برخی از معیارهای فاصله هزینه هر عملیات غیر کپی (از جمله جایگزینی) ۱ است.

ME distance - example

- distance("William Cohen", "William Cohon")

<i>s</i>	W	I	L	L	gap	I	A	M	_	C	O	H	E	N
							alignment							
<i>t</i>	W	I	L	L	L	I	A	M	_	C	O	H	O	N
<i>edit op</i>	C	C	C	C	I	C	C	C	C	C	C	C	S	C
<i>cost</i>	0	0	0	0	1	1	1	1	1	1	1	1	2	2
<i>so far...</i>									

Minimum Edit Distance (Levenshtein)

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 5
- If substitutions cost 2 (Levenshtein)
 - Distance between these is 8

Min Edit Example

	i	n	t	e	n	t	i	o	n			
delete i →		n	t	e	n	t	i	o	n			
substitute n by e →			e	t	e	n	t	i	o	n		
substitute t by x →				x	e	n	t	i	o	n		
insert u →					u	t	i	o	n			
substitute n by c →							c	u	t	i	o	n

Min Edit As Search

- But that generates a huge search space
 - Initial state is the word we're transforming
 - Operators are copy, insert, delete, substitute
 - Goal state is the word we're trying to get to
 - Path cost is what we're trying to minimize
- Search paths would be incredibly wasteful
- Why?

Lots of paths wind up at the same states. But there's no need to keep track of them all. We only care about the shortest path to each of those revisited states.

Min Edit Distance

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

$n \leftarrow \text{LENGTH}(\text{target})$

$m \leftarrow \text{LENGTH}(\text{source})$

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

Initialize the zeroth row and column to be the distance from the empty string

$\text{distance}[0,0] = 0$

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

$\text{distance}[i,0] \leftarrow \text{distance}[i-1,0] + \text{ins-cost}(\text{target}[i])$

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

$\text{distance}[0,j] \leftarrow \text{distance}[0,j-1] + \text{del-cost}(\text{source}[j])$

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

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

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

return $\text{distance}[n,m]$

Dynamic Program Table for String Edit

Measure distance between strings

PARK

SPAKE

		P	A	R	K
S					
P					
A			c_{ij}		
K					
E					

c_{ij} =
the number of edit
operations needed
to align PA with
SPA.

Dynamic Program Table for String Edit

		P	A	R	K
	c_{00}	c_{02}	c_{03}	c_{04}	c_{05}
S	c_{10}	c_{11}	c_{12}	c_{13}	c_{14}
P	c_{20}	c_{21}	c_{22}	c_{23}	c_{24}
A	c_{30}	c_{31}	???		
K					
E					

subst

delete

insert

$D(i,j)$ = score of **best** alignment from $s1..si$ to $t1..tj$

$$= \min \begin{cases} D(i-1,j-1), & \text{if } s_i=t_j & //copy \\ D(i-1,j-1)+1, & \text{if } s_i \neq t_j & //substitute \\ D(i-1,j)+1 & & //insert \\ D(i,j-1)+1 & & //delete \end{cases}$$

Dynamic Program Table Initialized

		P	A	R	K
	0	1	2	3	4
S	1	?			
P	2				
A	3				
K	4				
E	5				

$D(i,j)$ = score of **best** alignment from $s1..si$ to $t1..tj$

$$= \min \begin{cases} D(i-1,j-1)+d(s_i,t_j) & //substitute \\ D(i-1,j)+1 & //insert \\ D(i,j-1)+1 & //delete \end{cases}$$

Dynamic Program Table... Filling in

		P	A	R	K
	0	1	2	3	4
S	1	1			
P	2				
A	3				
K	4				
E	5				

$D(i,j)$ = score of **best** alignment from $s1..si$ to $t1..tj$

$$= \min \begin{cases} D(i-1,j-1)+d(s_i,t_j) & // \text{substitute} \\ D(i-1,j)+1 & // \text{insert} \\ D(i,j-1)+1 & // \text{delete} \end{cases}$$

Dynamic Program Table ... filling in

		P	A	R	K
	0	1	2	3	4
S	1	1	2	3	4
P	2	?			
A	3				
K	4				
E	5				

$D(i,j)$ = score of **best** alignment from $s1..si$ to $t1..tj$

$$= \min \begin{cases} D(i-1,j-1)+d(s_i,t_j) & //substitute \\ D(i-1,j)+1 & //insert \\ D(i,j-1)+1 & //delete \end{cases}$$

		P	A	R	K
	0	1	2	3	4
S	1	1	2	3	4
P	2	1			
A	3				
K	4				
E	5				

$D(i,j)$ = score of **best** alignment from $s1..si$ to $t1..tj$

$$= \min \begin{cases} D(i-1,j-1)+d(s_i,t_j) & //substitute \\ D(i-1,j)+1 & //insert \\ D(i,j-1)+1 & //delete \end{cases}$$

Dynamic Program Table ... filling in

		P	A	R	K
	0	1	2	3	4
S	1	1	2	3	4
P	2	1	2	3	4
A	3	2	1	2	3
K	4	3	2	2	2
E	5	4	3	3	3

Final cost of aligning all of both strings.

$D(i,j)$ = score of **best** alignment from $s1..si$ to $t1..tj$

$$= \min \begin{cases} D(i-1,j-1)+d(s_i,t_j) & // \text{substitute} \\ D(i-1,j)+1 & // \text{insert} \\ D(i,j-1)+1 & // \text{delete} \end{cases}$$

MED levenstein

0	λ	P	A	R	K
λ	0	1 \leftarrow	2 \leftarrow	3 \leftarrow	4 \leftarrow
S	1 \uparrow	2 $\uparrow \leftarrow \leftrightarrow$	3 $\uparrow \leftarrow \leftrightarrow$	4 $\uparrow \leftarrow \leftrightarrow$	5 $\uparrow \leftarrow \leftrightarrow$
P	2 \uparrow	1 \leftrightarrow	2 \leftarrow	3 \leftarrow	4 \leftarrow
A	3 \uparrow	2 \uparrow	1 \leftrightarrow	2 \leftarrow	3 \leftarrow
K	4 \uparrow	3 \uparrow	2 \uparrow	3 $\uparrow \leftarrow \leftrightarrow$	2 \leftrightarrow
E	5 \uparrow	4 \uparrow	3 \uparrow	4 $\uparrow \leftarrow \leftrightarrow$	3 \uparrow

Remebering the Alignment (trace)

$$D(i,j) = \min \begin{cases} D(i-1,j-1) + d(s_i,t_j) & //subst/copy \\ D(i-1,j)+1 & //insert \\ D(i,j-1)+1 & //delete \end{cases}$$

A *trace* indicates where the min value came from, and can be used to find edit operations and/or a best *alignment* (may be more than 1)

	C	O	H	E	N
M	1	2	3	4	5
C	1↑	2	3	4	5
C	2↑	3	3	4	5
O	3	2↖	3	4	5
H	4	3	2↖	3	4
N	5	4	3	3←	3↖

	#	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

		G	A	G	C	T	A
	0	1	2	3	4	5	6
A	1	2	1	2	3	4	5
A	2	3	2	3	4	5	4
C	3	4	3	4	3	4	5
G	4	3	4	3	4	5	6
C	5	4	5	4	3	4	5
A	6	5	4	5	4	5	4