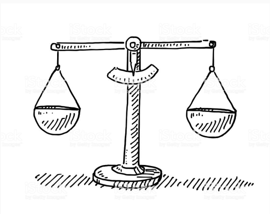
Edit Distance

Guillaume Wisniewski
guillaume.wisniewski@linguist.univ-paris-diderot.fr
September 2019
Université de Paris & LLF

Definition

How similar are two strings?



distance between two strings

- ↪ how similar are `dad` and `daddy`?
- ↪ is `tommorrow` closer to `tomorrow` or to `tomorow`?

Edit distance

String transformation

- given an alphabet Σ and a word $w \in \Sigma^*$
- Editing operations
  - ↪ insert the character x at position i
  - ↪ delete the character at position i
  - ↪ substitute the character at position i by x
- a sequence of operations will transform a word w into w'

Edit distance

- given two strings u, v, consider the set of all sequence of editions
- define a cost for each operation (usually 1)
- distance(u, v) = minimum number of operations to transform u into v

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

- ↪ no shorter sequence of operations
- ↪ edit distance between `intention` and `execution` is 5
- ↪ Levenshtein distance ($\omega_{\text{subst}} = 2$): 8

Alignment

- the sequence of edit operations define an alignment between the two strings

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

Is the edit distance a distance?

If:

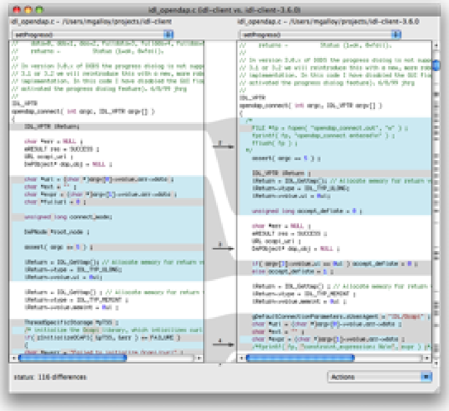
- every edit operation has positive cost;
- for every operation, there is an inverse operation with equal cost.

The metric axioms are satisfied as follows:

- $d(a, b) = 0$ if and only if $a = b$, each string is trivially transformed to itself with a cost of 0
- $d(a, b) > 0$ when $a \neq b$, at least one edit operation
- $d(a, b) = d(b, a)$ by equality of the cost of each operation and its inverse.
- Triangle inequality: $d(a, c) \leq d(a, b) + d(b, c)$

Application

Comparing documents



Identifiying Frequent Spelling Correction in French

[Wisniewski et al, TALN'2010]

- compute alignement between the different revision of a wikipedia page
- heuristics to filter modifications that correspond to spelling errors
- compute statistics

Erreurs orthographiques

e → é	6,7%	-l	1,9%
E → É	6,7%	+i	1,9%
oe → œ	4,6%	a → â	1,8%
+n	4,3%	-e	1,7%
+s	2,8%	-n	1,7%
+r	2,7%	+t	1,6%
é → è	2,7%	+m	1,6%
-s	2,5%	e → è	1,4%
+e	2,2%	+l	1,3%
é → e	2,1%	-r	1,3%

Erreurs grammaticales

+s	16.2%	-t	1.5%
+e	9.9%	e → a	1.4%
-s	8.8%	é → er	1.0%
A → À	5.6%	er → é	0.9%
-e	4.9%	u → ù	0.9%
i → î	2.7%	à → a	0.9%
a → à	2.2%	e → é	0.8%
+nt	1.9%	é → è	0.7%
+t	1.7%	s → t	0.7%
a → e	1.5%	û → u	0.7%

Identifiying Frequent Spelling Correction in French

[Wisniewski et al, TALN'2010]

- compute alignement between the different revision of a wikipedia page
- heuristics to filter modifications that correspond to spelling errors
- compute statistics

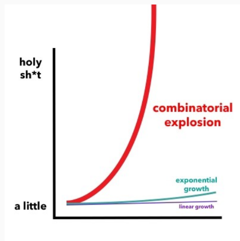
	erreurs lexicales	erreurs grammaticales
première moitié du mot	34,06%	4,08%
seconde moitié du mot	62,81%	93,26%
erreurs dans les deux moitiés	3,13%	2,63%

Computing minimal edit distance

Number of sequence of edits

Given two strings a and b, how many sequence of edits can transform a into b?

Number of sequence of edits



Observation

- combinatorial explosion
- number of answers grows very/too quickly with respect to the size of the words

Consequence

- minimum edit is a search problem can be reduced to shortest path problem
- the space of all edit sequences is huge!
- we can afford to navigate

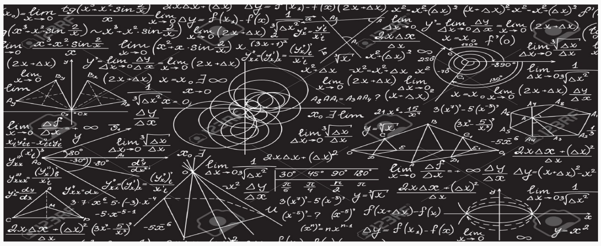
Some observations

1. lots of different sequence of editions result in the same word:
   - a can be transformed into b by:
   - ↪ 1 substitution
   - ↪ 1 deletion and 1 insertion
   - ↪ ....

   ⇒ overlapping sub-problems
2. minimal edit distance → only need to keep track of the 'shortest' way to generate a given word

⇒ optimal substructure

Some notations



- X, Y: two strings of length n and m
- $D(i, j)$: min. edit distance between $X[1..i]$ and $Y[1..j]$
  - ↪ the first i characters of X and j characters of Y
  - ↪ the 'overlapping sub-problems'
- $D(n, m)$ is the edit distance between X and Y

Dynamic Programming

- dynamic programming: a tabular computation of $D(n, m)$
- solving problems by combining solutions to sub-problems
- bottom-up:
  - ↪ compute $D(i, j)$ for 'small' i, j
  - ↪ compute 'larger' $D(i, j)$ based on previously computed smaller values
- avoid computing the same 'sub-result' twice

An historical wink (1)

- dynamic programming (Bellman, 1950) = generic optimization method
- many applications (e.g. shortest path problem, Viterbi algorithm, matrix multiplication, ...)
- explore an exponential number of solutions in a polynomial time
  - ↪ simply by 'organizing' the computation correctly

An historical wink (2)

"Where did the name, dynamic programming, come from? The 1950s were not good years for mathematical research. We had a very interesting gentleman in Washington named Wilson. He was Secretary of Defense, and he actually had a pathological fear and hatred of the word research. [...] I decided therefore to use the word "programming". I wanted to get across the idea that this was dynamic, this was multistage, this was time-varying. I thought, let's [...] take a word that has an absolutely precise meaning, namely dynamic [...] it's impossible to use the word dynamic in a pejorative sense. Try thinking of some combination that will possibly give it a pejorative meaning. It's impossible. Thus, I thought dynamic programming was a good name. It was something not even a Congressman could object to. So I used it as an umbrella for my activities."

Richard Bellman, "Eye of the Hurricane: an autobiography" 1984.

Computing D (1)

Obviously:

$$\begin{cases} D(i,0) = i & \forall i \in [\![1,n]\!] \\ D(0,j) = j & \forall j \in [\![1,m]\!] \end{cases} \tag{1}$$

- $\hookrightarrow$ $D(i,0) \rightarrow$ generate the empty string from the first i character of X
- $\hookrightarrow$ 'best' way: i deletions
- $\hookrightarrow$ recursive definition: $D(i,0) = D(i-1,0)+1 \rightarrow$ considering a larger suffix, require one extra deletion

Computing D (2)

$$D(i,j) = \begin{cases} D(i-1,j-1) & \text{if } X[i] = Y[j] \\ \min \begin{cases} d[i-1,j] + \omega_{\text{del}}(X[i]) \\ d[i,j-1] + \omega_{\text{ins}}(Y[j]) \\ d[i-1,j-1] + \omega_{\text{sub}}(X[i], Y[j]) \end{cases} & \text{if } X[i] \neq Y[j] \end{cases} \tag{2}$$

- $\hookrightarrow$ compute the edit distance incrementally
- $\hookrightarrow$ at each step: best way to 'extend' the best solution achieved so far

For the moment...

$$\begin{cases} \omega_{\text{del}}(a) = 1 & \forall a \in \Sigma \\ \omega_{\text{sub}}(b) = 1 & \forall b \in \Sigma \\ \omega_{\text{ins}}(a,b) = 2 & \forall a,b \in \Sigma^2 \end{cases}$$

The edit distance table

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

The edit distance table

N										
O										
I										
T										
N										
E										
T										
N										
I										
#	0									
	#	E	X	E	C	U	T	I	O	N

The edit distance table

N										
O										
I										
T										
N										
E										
T										
N										
I	1									
#	0									
	#	E	X	E	C	U	T	I	O	N

The edit distance table

N										
O										
I										
T										
N										
E										
T										
N	2									
I	1									
#	0									
	#	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									
	#	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

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

$$X[i] \neq X[j] \Rightarrow \min \begin{cases} 1+1 & \text{insertion} \\ 1+1 & \text{deletion} \\ 0+2 & \text{substitution} \end{cases}$$

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	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

Complexity

- time complexity: $\mathcal{O}(n \times m)$
- space complexity;: $\mathcal{O}(n \times m)$

Are we done yet?

- efficient computation of the minimal edit distance value
- how can recover the corresponding sequence of editions?

Are we done yet?

- efficient computation of the minimal edit distance value
- how can recover the corresponding sequence of editions?
  $\Rightarrow$ backtracking

Backtracking

- for each entry in the edit distance table: keep track of the cell we came from
  i.e. the operation corresponding to the min

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**

Weighted edit distance

Why would we add weights to the computation?

$\hookrightarrow$ e.g. different weights for different substitutions

Weighted edit distance

Spell checkers: some errors are more probable than other (even worst for T9)

Weighted edit distance

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 edit distance

↪ use distance between letters on a keyboard