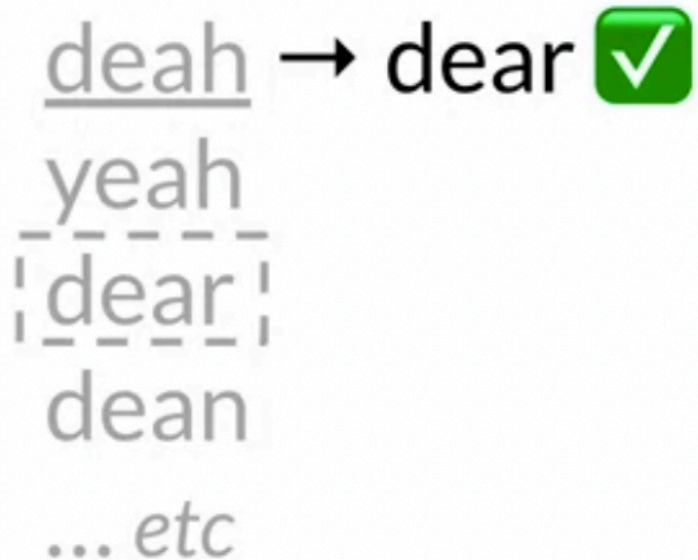
What is autocorrect?

- Example:

Happy birthday deer friend!  ??

How it works

1. Identify a misspelled word
2. Find strings n edit distance away
3. Filter candidates
4. Calculate word probabilities



Building the model

1. Identify a misspelled word

```
if word not in vocab:
    misspelled = True
```

deah ❌

deer 🦌 ✅

Building the model

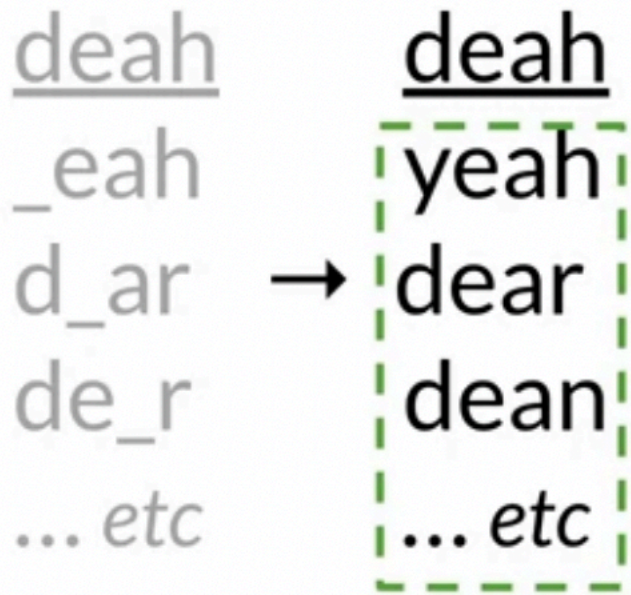
2. Find strings n edit distance away

- Edit: an operation performed on a string to change it

- Insert (add a letter) 'to': 'top', 'two' ...
- Delete (remove a letter) 'hat': 'ha', 'at', 'ht'
- Switch (swap 2 adjacent letters) 'eta': 'eat', 'tea'
- Replace (change 1 letter to another) 'jaw': 'jar', 'paw', ...

Building the model

3. Filter candidates



Building the model

4. Calculate word probabilities

Example: "I am happy because I am learning"

$$P(w) = \frac{C(w)}{V}$$

$$P(\text{am}) = \frac{C(\text{am})}{V} = \frac{2}{7}$$

$P(w)$ Probability of a word

$C(w)$ Number of times the word appears

V Total size of the corpus

Word	Count
I	2
am	2
happy	1
because	1
learning	1

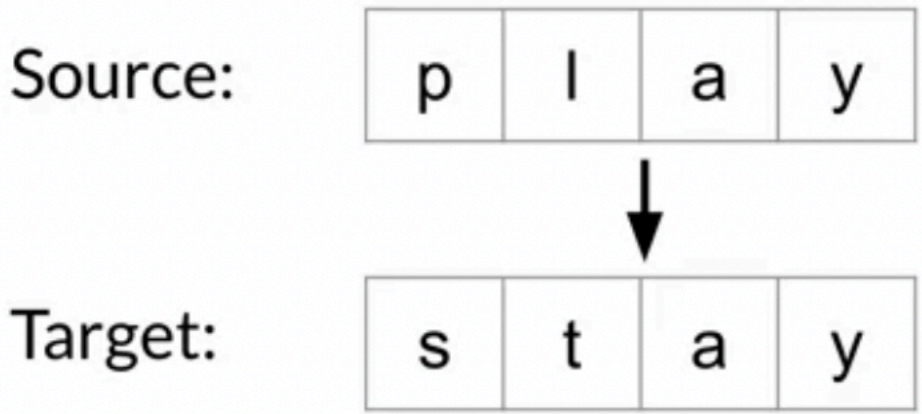
Total : 7

Minimum edit distance

- How to evaluate similarity between 2 strings?
- Minimum number of edits needed to transform 1 string into the other
- Spelling correction, document similarity, machine translation, DNA sequencing, and more

Minimum edit distance

- Example:



Edit cost:
Insert 1
Delete 1
Replace 2

p → s : replace
l → t : replace

edits = 2

Minimum edit distance

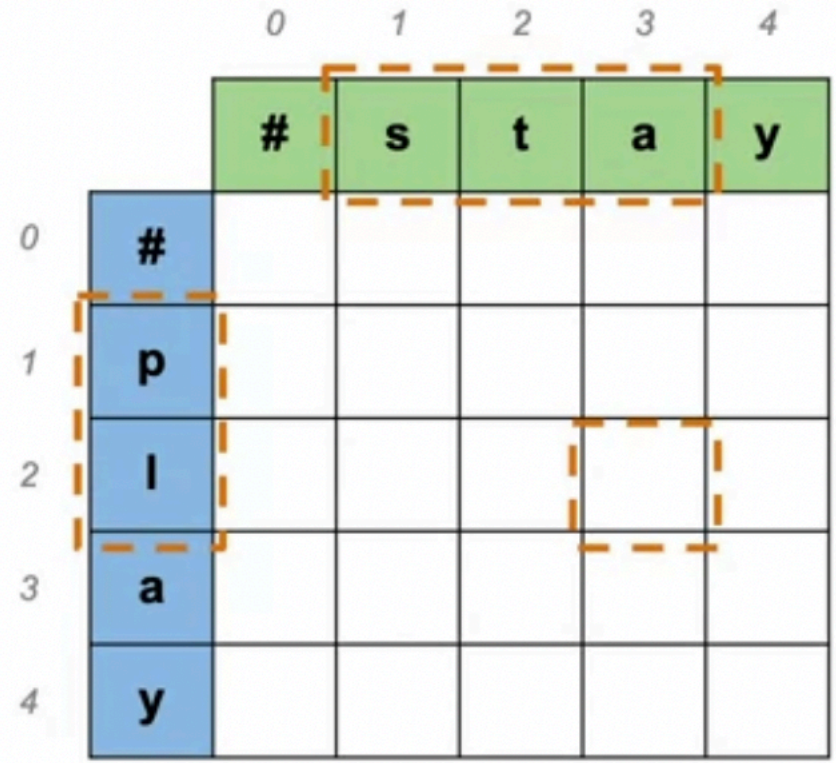
Source: play → Target: stay

$D[\ ]$

$D[2,3]$ = pl → sta

$D[2,3]$ = source[:2] → target[:3]

$D[i,j]$ = source[: i] → target[: j]



Minimum edit distance

Source: play → Target: stay

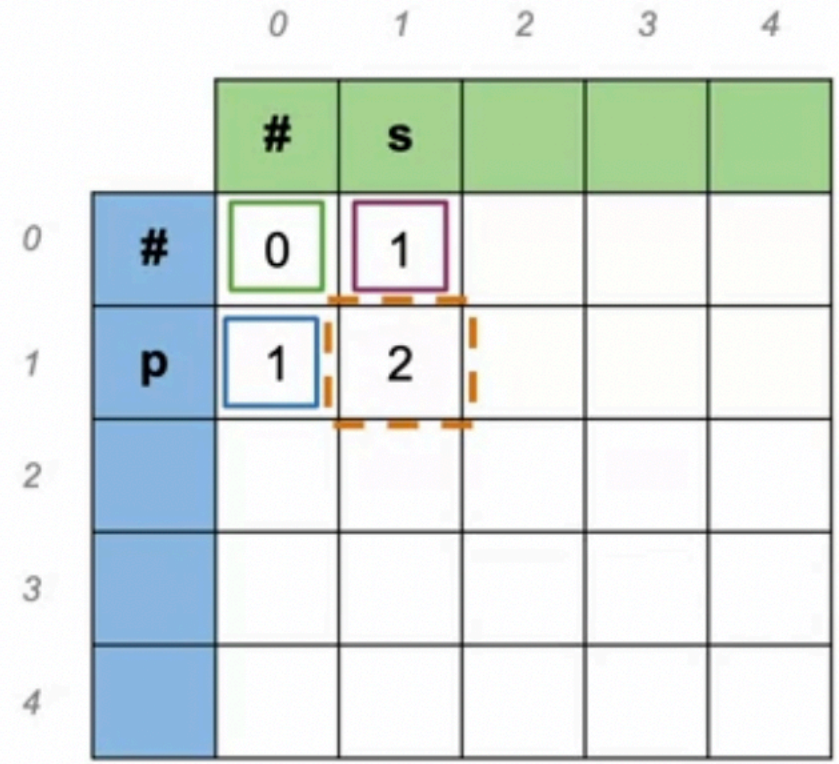
Cost: insert: 1, delete: 1, replace: 2

p → s

insert + *delete*: p → ps → s: 2

delete + *insert*: p → # → s: 2

replace: p → s: 2



Minimum edit distance

Source: play → Target: stay

Cost: insert: 1, delete: 1, replace: 2

play → #

$D[i,j] = D[i\text{-}1,j] + del\_cost$

$D[4,0]$ = play → #

= source[:4] → target[0]

		0	1	2	3	4
		#	s	t	a	y
0	#	0	1			
1	p	1	2			
2	l	2				
3	a	3				
4	y	4				

Minimum edit distance

Source: play → Target: stay

Cost: insert: 1, delete: 1, replace: 2

p → s

$$D[i,j] = \min \begin{cases} D[i-1,j] + del\_cost \\ D[i,j-1] + ins\_cost \\ D[i-1,j-1] + \begin{cases} rep\_cost; & \text{if } src[i] \neq tar[j] \\ 0; & \text{if } src[i] = tar[j] \end{cases} \end{cases}$$

		0	1	2	3	4
		#	s	t	a	y
0	#	0	1	2	3	4
1	p	1	2			
2	l	2				
3	a	3				
4	y	4				

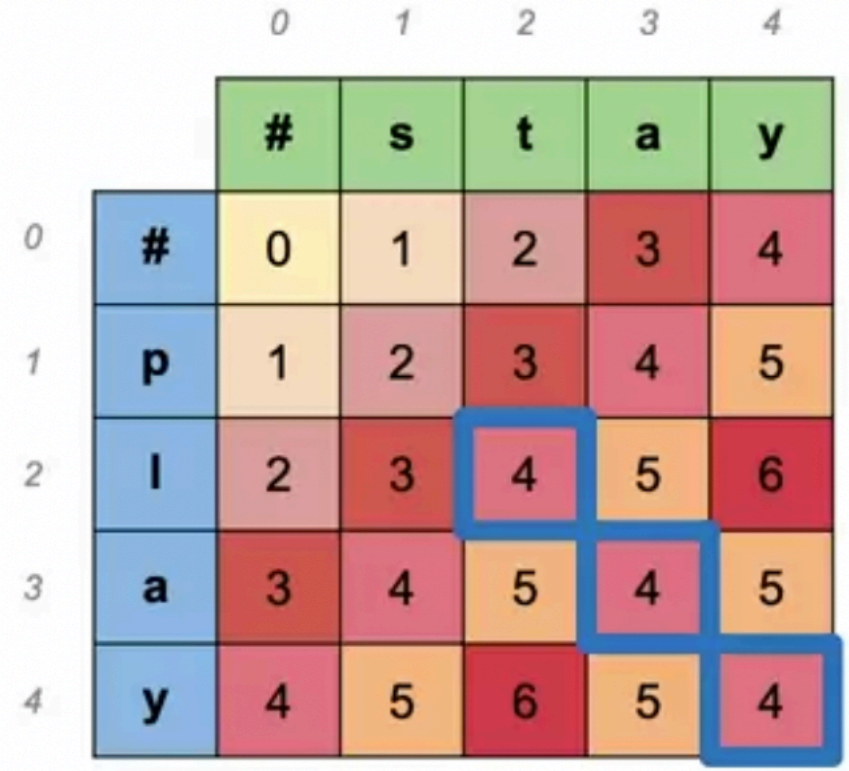
Minimum edit distance

Source: play → Target: stay

Cost: insert: 1, delete: 1, replace: 2

play → stay

D[*m*, *n*] = 4



		0	1	2	3	4
		#	s	t	a	y
0	#	0	1	2	3	4
1	p	1	2	3	4	5
2	l	2	3	4	5	6
3	a	3	4	5	4	5
4	y	4	5	6	5	4

Minimum edit distance

Source: play → Target: stay

Cost: insert: 1, delete: 1, replace: 2

- Levenshtein distance
- Backtrace
- Dynamic programming

		0	1	2	3	4
		#	s	t	a	y
0	#	0	1	2	3	4
1	p	1	2	3	4	5
2	l	2	3	4	5	6
3	a	3	4	5	4	5
4	y	4	5	6	5	4