

# 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

deah → dear 

yeah

[dear]

dean

... etc



# Building the model

1. Identify a misspelled word

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

deah



deer



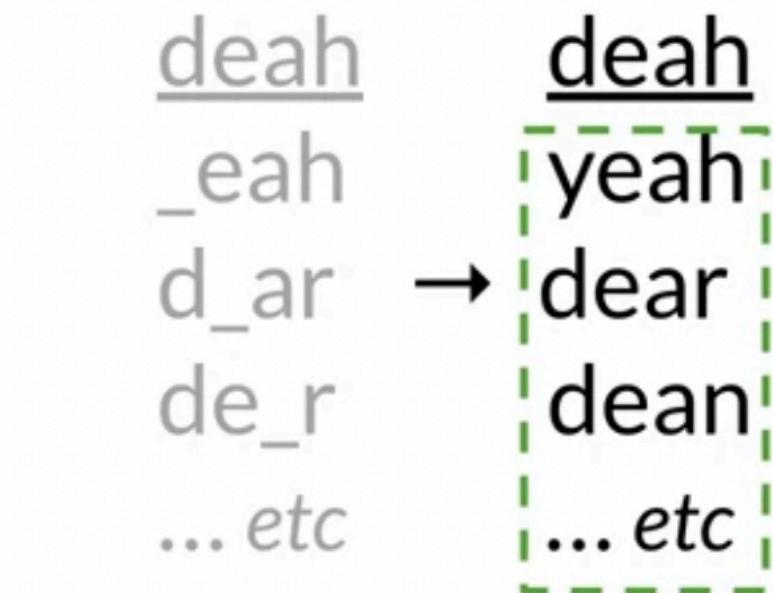
# Building the model

## 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:

Source:

p		a	y
---	--	---	---



Target:

s	t	a	y
---	---	---	---

Edit cost:  
Insert 1  
Delete 1  
Replace 2

p → s : replace  
l → t : replace

} edits = 2

# Minimum edit distance

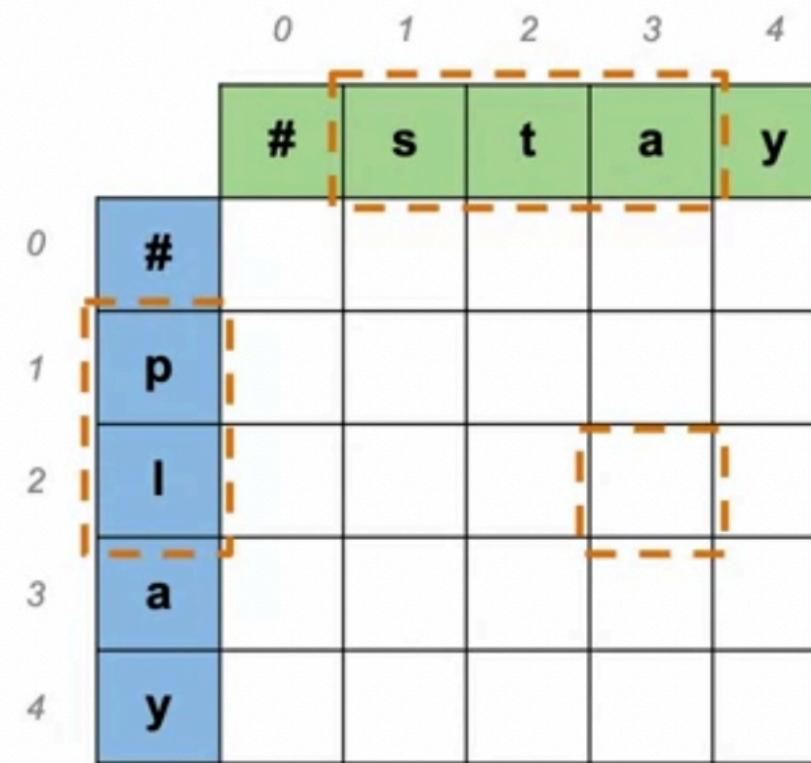
Source: play → Target: stay

$D[ ]$

$D[2,3] = \text{pl} \rightarrow \text{sta}$

$D[2,3] = \text{source}[ : 2] \rightarrow \text{target}[ : 3]$

$D[ i, j ] = \text{source}[ : i] \rightarrow \text{target}[ : j]$



# Minimum edit distance

Source: play → Target: stay

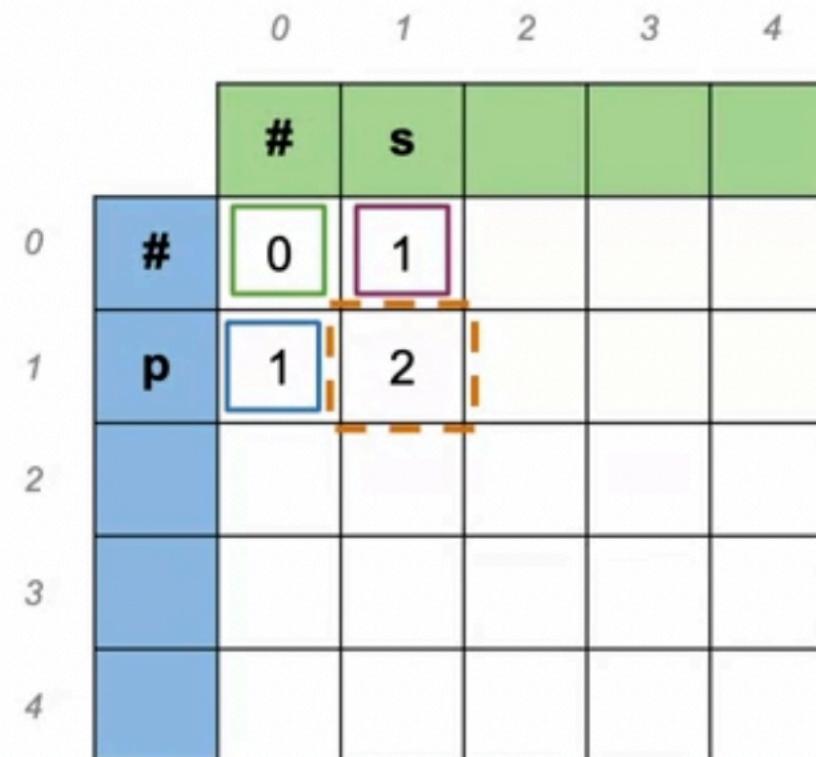
Cost: insert: 1, delete: 1, replace: 2

$p \rightarrow s$

**insert**+**delete**:  $p \rightarrow ps \rightarrow s$ : 2

**delete**+**insert**:  $p \rightarrow \# \rightarrow s$ : 2

**replace**:  $p \rightarrow s$ : 2



# Minimum edit distance

Source: play → Target: stay

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

play → #

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

$$\begin{aligned} D[4,0] &= \text{play} \rightarrow \# \\ &= \text{source}[ :4] \rightarrow \text{target}[0] \end{aligned}$$

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

An orange dashed line highlights the path from the bottom-left cell (y, 4) to the top-left cell (#, 0), indicating the sequence of edits required to transform "play" into "#".

# Minimum edit distance

Source: play → Target: stay

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

$p \rightarrow s$

$D[i, j] =$

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

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

The table shows the minimum edit distance matrix for transforming "play" into "stay". The rows represent the source string "play" and the columns represent the target string "stay". The cost of each edit operation is: insert: 1, delete: 1, replace: 2. The value at  $D[4, 4]$  is 4. A blue double-headed arrow highlights the path from the start to the end of the target string "stay".

# Minimum edit distance

Source: play → Target: stay

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

- Levenshtein distance
- Backtrace
- Dynamic programming

	0	1	2	3	4	
0	#	0	1	2	3	4
1	p	1	2	3	4	5
2	I	2	3	4	5	6
3	a	3	4	5	4	5
4	y	4	5	6	5	4