Natural Language Processing with Probabilistic Model

- Autocorrect
- Part of Speech Tagging and Hidden Markov Models
- Autocomplete and Language Models
- Word embeddings with neural networks

Autocorrect

- Learning objectives
  - What is autocorrect?
  - Building the model
  - Minimum edit distance
  - Minimum edit distance algorithm

deah → dear ✅
yeah
dear
dean
... *etc*

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

What is autocorrect?

Happy birthday deah friend!

Happy birthday deer friend! ??

Happy birthday dear friend!

Autocorrect is an application that changes misspelled words into the correct ones.

FPT Education

FPT UNIVERSITY

How it works

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

<u>deah</u>

<u>deah</u>
_eah
d_ar
de_r
... *etc*

<u>deah</u>
yeah
dear
dean
... *etc*

<u>deah</u>
yeah
dear
dean
... *etc*

Building the model

FPT Education

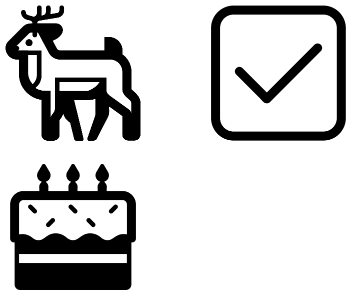
FPT UNIVERSITY

1. Identify a misspelled word

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

<u>deah</u>

Happy birthday deer !



- If word not in vocabulary then its misspelled
- More sophisticated techniques for identifying words that are probably incorrect by looking at the words surrounding them

Building the model

2. Find strings n edit distance away: find all possible strings that are n edit distance away using

- Edit: an operation performed on a string to change it
  - how many operations away one string is from another
    - Insert (add a letter)
      - Add a letter to a string at any position: to ==> top, two,...
    - Delete (remove a letter)
      - Remove a letter from a string : hat ==> ha, at, ht
    - Switch (swap 2 adjacent letters)
      - Example: eta=> eat, tea
    - Replace (change 1 letter to another): Example: jaw ==> jar,paw,saw,...
- By combining the 4 edit operations, we get list of all possible strings that are n edit

<u>deah</u>

_eah
d_ar
de_r
... *etc*

Building the model

3. Filter candidates

- From the list from step 2, consider only real and correct spelled word
- if the edit word not in vocabulary ==> remove it from list of candidates

deah		deah
_eah		yeah
d_ar	→	dear
de_r		dean
… *etc*		… *etc*

Building the model

- Calculate word probabilities: the word candidate is the one with the highest probabilitya word probablity in a corpus is: number of times the word appears divided by the total number of words.

Example: "I am happy because I am learning"

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

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

$P(w)$ 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 to the other
- Applications
  - Spelling correction,
  - Document similarity,
  - Machine translation,
  - DNA sequencing, and more

Minimum edit distance

- Example:

Source:

p	l	a	y

↓

Source:

s	t	a	y

$p \rightarrow s$: replace
$l \rightarrow t$: replace

edits = 2

Edit cost:

Insert	1
Delete	1
Replace	2

edit distance = 2 * 2 = 4

Minimum edit distance algorithm

- The source word layed on the column
- The target word layed on the row
- Empty string at the start of each word at (0,0)
- D[i,j] is the minimum editing distance between the beginning of the source word to index i and the beginning of the target word to index

Minimum edit distance

- Source: play → Target: stay
  - Have the source wordplay here down the left column
  - Targets were to transform into stay along the top row
  - The goal is to fill out this distance matrix D[ ]
  - D[i, j] = source[: i] target [:j]
  - D[m, n] = source → target

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

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

		0	1	2	3	4
		#	s			
0	#	0	1			
1	p	1	2			
2						
3						
4						

Minimum edit distance II

Source: play → Target: stay
Cost: insert: 1, delete: 1, replace: 2

play → #

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

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

		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				

		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				

- D[4,0], you have the minimum edit distance for play to the empty string
- same idea in the first row by transforming the empty string to stay by inserting one letter at a time

Minimum edit distance

Source: play $\rightarrow$ Target: stay

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

p $\rightarrow$ 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

		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

- The Levenshtein distance between two sequences is the simplest weighting factor in which each of the three operations has a cost of 1 (Levenshtein, 1966)—we assume that the substitution of a letter for itself, for example, t for t, has zero cost.

Minimum edit distance III

Source: play → Target: stay
Cost: insert: 1, delete: 1, replace: 2

- Levenshtein distance
- Backtrace: a pointer in each cell letting you know where character came from

		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:
  - solving the smallest subproblem first and then reusing that result to solve the next biggest subproblem

		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

Summary

- What is autocorrect?
- Building the model
- Minimum edit distance
- Minimum edit distance algorithm

		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