NLP is AI-hard (AI-Complete)

Let’s do it!

TEXT SEGMENTATION

dividing written text into meaningful units, such as words, sentences, or topics

Word Segmentation: Tokenization

- Whitespace (default, natural word delimiter)
- Exceptions

  New York
  rock 'n' roll
  Contractions: I'm
  Japanese | Chinese | Thai don't have spaces between words
  Emoticons: :)
  Hashtags: #nlproc.

Word Boundaries: Tokenization: Space

- Split()
- Regular Expressions (RE): Finite State Automata
  - Alphabetical: [a-zA-Z]*
  - Alpha-numerical: [a-zA-Z0-9]*
  - Punctuations: Ph.D., AT&T, cap'n
  - Special Chars

    Currency $45.55
    Dates (01/02/06)
    URLs http://www.stanford.edu
    Twitter hashtags #nlproc
    Email hfani@uwindsor.ca

What should be considered as word?

- Disfluencies in *utterances*
  - Fragments: broken-off repeated words: miss- misspelled, you- yourself
  - Fillers: non-lexical: huh, uh, erm, um, well, so, like, hmm
- Punctuations , . : ; ? !
  - part-of-speech tagging
  - parsing
  - speech synthesis
- Morphemes:
  - smallest meaning-bearing unit of a language
  - 'unlikeliest' : morphemes [un-], [likely], [-est]

What should be considered as word?

- Chinese

As Chen et al. (2017) point out, this could be treated as 3 words ('Chinese Treebank' segmentation):

(2.5) 姚明 进入 总决赛
YaoMing reaches finals

or as 5 words ('Peking University' segmentation):

(2.6) 姚 明 进入 总 决赛
Yao Ming reaches overall finals

Finally, it is possible in Chinese simply to ignore words altogether and use characters as the basic elements, treating the sentence as a series of 7 characters:

(2.7) 姚 明 进 入 总 决 赛
Yao Ming enter enter overall decision game

Chen, X., Shi, Z., Qiu, X., and Huang, X. (2017). Adversarial multi-criteria learning for Chinese word segmentation. In ACL 2017, 1193–1203.

What should be considered as word?

- Chinese

characters are at a reasonable semantic level for most applications
most word standards result in a huge vocabulary with large numbers of very rare words
Take characters as words

As Chen et al. (2017) point out, this could be treated as 3 words ('Chinese Treebank' segmentation):

(2.5) 姚明 进入 总决赛
YaoMing reaches finals

or as 5 words ('Peking University' segmentation):

(2.6) 姚 明 进入 总 决赛
Yao Ming reaches overall finals

Finally, it is possible in Chinese simply to ignore words altogether and use characters as the basic elements, treating the sentence as a series of 7 characters:

(2.7) 姚 明 进 入 总 决 赛
Yao Ming enter enter overall decision game

LEARN TO TOKENIZE

Word→ Subword → Word

- Byte-Pair Encoding (BPE)

Sennrich, et al., (2016). Neural machine translation of rare words with subword units. In ACL 2016.

- Wordpiece

Wu et al. (2016) Google's neural machine translation system: Bridging the gap between human and machine translation." arXiv.

- MaxMatch in BERT

Devlin et al. (2019). BERT: Pretraining of deep bidirectional transformers for language understanding. In NAACL HLT.

- SentencePiece

Kudo, T. and Richardson, J. (2018). SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In EMNLP.

Word→ Subword → Word

- Byte-Pair Encoding (BPE)

Sennrich, et al., (2016). Neural machine translation of rare words with subword units. In ACL 2016.

1. Rare words into subword units is sufficient for translation
2. Help to generalize to translate and produce *unseen* words

100 rare tokens in German training data and they are translatable from English via smaller units!

https://github.com/rsennrich/subword-nmt

Word→ Subword → Word

- Byte-Pair Encoding (BPE)

Sennrich, et al., (2016). Neural machine translation of rare words with subword units. In ACL 2016.

Data Compression (Gage, P. (1994). A new algorithm for data compression. The C Users Journal, 12(2), 23–38.)

aaabdaaabac → *pair 'aa' occurs most often* → *replace it with a char (byte) that is not used 'Z'*

→ ZabdZabac; Z=aa

→ *pair 'ab' occurs most often* → *replace it with 'Y'*

→ ZYdZYac; Y=ab, Z=aa

→ *byte pair 'ZY' with 'X'*

→ XdXac; X=ZY, Y=ab, Z=aa

This data cannot be compressed further by byte pair encoding because there are no pairs of bytes that occur more than 1.

Word→ Subword → Word

- Byte-Pair Encoding (BPE)

Sennrich, et al., (2016). Neural machine translation of rare words with subword units. In ACL 2016.

1- Initialization a Dictionary: Tokenize Text or Input an Available One
2- Initialize a Vocabulary: {unique characters} U {end-of-word symbol=</w>}
Loop:
3- get_stats((x, y) ∈ Vocabulary)
4- (A, B) = most frequent pair
4- Vocabulary U {Add most frequent pair='AB'}
5- Dictionary.replace('A B', 'AB')

Word→ Subword → Word

- Byte-Pair Encoding (BPE)

Dictionary$^{(0)}$

5: 'l o w </w>'
2: 'l o w e r </w>'
6: 'n e w e s t </w>'
3: 'w i d e s t </w>'

Vocabulary$^{(0)}$

l, o, w, e, r, t, n, s, d, </w>

get_stats()
0: (l, l)
7: (l, o)
0: (l, w)
....
0: (o, l)
0: (o, o)
...
9: (e, s)
0: (o, </w>)
....

(e, s) = most frequent pair

Word→ Subword → Word

- Byte-Pair Encoding (BPE)

Dictionary[1]

5: 'l o w </w>'
2: 'l o w e r </w>'
6: 'n e w es t </w>'
3: 'w i d es t </w>'

Vocabulary[1]

l, o, w, e, r, t, n, s, d, </w>,
es,

get_stats()
0: (l, l)
7: (l, o)
0: (l, w)
....
0: (o, l)
0: (o, o)
...
0: (e, s)
9: (es, t)
....

(es, t) = most frequent pair

Word→ Subword → Word

- Byte-Pair Encoding (BPE)

Dictionary(2)

5: 'l o w </w>'
2: 'l o w e r </w>'
6: 'n e w est </w>'
3: 'w i d est </w>'

Vocabulary(2)

l, o, w, e, r, t, n, s, d, </w>,
es, est,

get_stats()
0: (l, l)
7: (l, o)
0: (l, w)
....
0: (o, l)
0: (o, o)
...
0: (e, s)
0: (es, t)
....
9: (est, </w>)

9: (est, </w>) = most frequent pair

Word→ Subword → Word

- Byte-Pair Encoding (BPE)

Dictionary(3)

5: 'l o w </w>'
2: 'l o w e r </w>'
6: 'n e w est</w>'
3: 'w i d est</w>'

Vocabulary(3)

l, o, w, e, r, t, n, s, d, </w>,
es, est, est</w>

get_stats()
0: (l, l)
7: (l, o)
0: (l, w)
....
0: (o, l)
0: (o, o)
...
0: (e, s)
0: (es, t)
....
0: (est, </w>)

9: (est, </w>) = most frequent pair

Word→ Subword → Word

- Byte-Pair Encoding (BPE)

Dictionary$^{(n)}$

5: ‘l o w </w>'
2: ‘l o w e r </w>'
6: 'n e w est</w>'
3: 'w i d est</w>'

Vocabulary$^{(n)}$

l, o, w, e, r, t, n, s, d, </w>,
es, est, est</w>, ...
newer
wider
lowest

Word→ Subword → Word

- Wordpiece

Wu et al. (2016) Google's neural machine translation system: Bridging the gap between human and machine translation." arXiv.

Same as BPE but:

- </w> appears at the beginning of words
- merging the pairs that minimizes the language model likelihood of the training data.

Word→ Subword → Word

- MaxMatch in BERT

Devlin et al. (2019). BERT: Pretraining of deep bidirectional transformers for language understanding. In NAACL HLT.

function MAXMATCH(string, dictionary) **returns** list of tokens T

    if string is empty
        return empty list
    for $i \leftarrow length$(sentence) **downto** 1
        firstword = first *i* chars of *sentence*
        remainder = rest of *sentence*
        if InDictionary(*firstword*, *dictionary*)
            return list(firstword, MaxMatch(remainder,dictionary))

Word→ Subword → Word

- MaxMatch in BERT

Devlin et al. (2019). BERT: Pretraining of deep bidirectional transformers for language understanding. In NAACL HLT.

- unaffable
  - [u][naffable]
  - [un][affable]
  - [una][ffable]
  - [unaf][fable]
  - [unaff][able]
  - [unaffa][ble]
  - [unaffab][le]
  - [unaffabl][e]
  - [unaffable]

```
function MAXMATCH(string, dictionary) returns list of tokens T

    if string is empty
        return empty list
    for i ← length(sentence) downto 1
        firstword = first i chars of sentence
        remainder = rest of sentence
        if InDictionary(firstword, dictionary)
            return list(firstword, MaxMatch(remainder,dictionary) )
```

Word→ Subword → Word

- MaxMatch in BERT

Devlin et al. (2019). BERT: Pretraining of deep bidirectional transformers for language understanding. In NAACL HLT.

- unaffable
  - [u][naffable]
    return
  - [un][affable]
  - [una][ffable]
  - [unaf][fable]
  - [unaff][able]
  - [unaffa][ble]
  - [unaffab][le]
  - [unaffabl][e]
  - [unaffable]

function MAXMATCH(string, dictionary) **returns** list of tokens T

    if string is empty
        return empty list
    for $i \leftarrow$ *length*(sentence) **downto** 1
        firstword = first *i* chars of *sentence*
        remainder = rest of *sentence*
        if InDictionary(*firstword*, *dictionary*)
            return list(firstword, MaxMatch(remainder,dictionary))

Word→ Subword → Word

- MaxMatch in BERT

Devlin et al. (2019). BERT: Pretraining of deep bidirectional transformers for language understanding. In NAACL HLT.

- unaffable
  - [u][naffable]
  - [un][affable]
    - {'un'} U [affable]
      - [a][ffable]
      - [af][fable]
      - [aff][able]
      - …
  - [una][ffable]
  - [unaf][fable]
  - [unaff][able]
  - [unaffa][ble]
  - [unaffab][le]

```
function MAXMATCH(string, dictionary) returns list of tokens T
    if string is empty
        return empty list
    for i ← length(sentence) downto 1
        firstword = first i chars of sentence
        remainder = rest of sentence
        if InDictionary(firstword, dictionary)
            return list(firstword, MaxMatch(remainder,dictionary) )
```

Word→ Subword → Word

- MaxMatch in BERT

Devlin et al. (2019). BERT: Pretraining of deep bidirectional transformers for language understanding. In NAACL HLT.

unaffable → [un, ##aff, ##able]
intention → [intent, ##ion]
unwanted running → [un, ##want, ##ed, runn, ##ing]

→ marking as internal subwords that do not start words

Word→ Subword → Word

- SentencePiece

Kudo, T. and Richardson, J. (2018). SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In EMNLP.

Whitespace is a character

For languages that space is not a delimiter

Sentence Segmentation

- Boundary markers
  - Exclamation (!)
  - Question (?)
  - Period (.)
    - Abbreviation: Mr. or Inc.
    - Both

      The conference was in ABS Inc.

Sentence Segmentation

- Rule-based: Regular Expression
  Stanford's CoreNLP
  Combined by word segmentation (Tokenizer)
- Learn to Segment
  - Learn to label (.) as sentence marker or abbreviation marker or both

TEXT NORMALIZATION

- Case Folding
  mapping everything to lower (upper) case
- Lemmatization
  converts the word to its meaningful base form, which is called Lemma. The same word may have multiple different Lemmas
- Stemming
  removes or stems the last few characters of a word, often leading to incorrect meanings and spelling
- Autocorrection

Case Folding

Positive Impact: ‘USA’ vs. ‘usa’

+ Information Retrieval
+ Speech Recognition

Negative Impact: ‘US’ the country vs. ‘us’ the pronoun

- Sentiment Analysis
- Text Classification
- Information Extraction
- Machine Translation

Lemmatization

- Polysemy

the association of one word with two or more distinct meanings
a polyseme is a word or phrase with multiple meanings

A polyseme may have multiple different Lemmas → Disambiguation

Saw as noun vs. Saw → See

Stemming: simple but crude lemmatization

- Mainly consists of chopping off word-final stemming affixes.
- Porter, M. F. (1980). An algorithm for suffix stripping. Program, 14(3), 130–137.

Errors of Commission		**Errors of Omission**	
organization	organ	European	Europe
doing	doe	analysis	analyzes
numerical	numerous	noise	noisy
policy	police	sparse	sparsity

A commission error is an error made due to using an item in the wrong context.

Spelling Correction via Minimum Edit Distance

Word Similarity (Distance)

- In surface 'Minimum' 'Maximum'
- In semantic 'Attention' 'Focus'
- In relatedness 'Company' 'Employee'

Levenshtein (1966)

Distance between two sequences is the total cost of changing one to reach the other one

Word Distance:

Insertion (cost = 1)

Deletion (cost = 1)

Substitution (Insertion + Deletion = 2)

Levenshtein, V. I. (1966). Binary codes capable of correcting deletions, insertions, and reversals. Cybernetics and Control Theory (1965).

Spelling Correction via Minimum Edit Distance

- Recursive Programing

Two words are similar if they are similar up until the last chars!

natura vs. nature

The distance of two words is the distance of their subwords till the last chars!

$$D(x[0:n], y[0:n]) = D(x[0:i-1], y[0:i-1]) + \text{the cost of } x[i:n] \leftrightarrow y[i:n]$$
$$i = n$$

Spelling Correction via Minimum Edit Distance

- Recursive Programing

$$D(x[0:n], y[0:n]) = D(x[0:i-1], y[0:i-1]) + \text{the cost of } x[i:n] \leftrightarrow y[i:n]$$
$$i = n$$

D([natura], D[nature])

D([natur], D[natur]) + [a] ↔ y[e]

Spelling Correction via Minimum Edit Distance

- Recursive Programing

$$D(x[0:n], y[0:n]) = D(x[0:i\text{-}1], y[0:i\text{-}1]) + \text{the cost of } x[i:n] \leftrightarrow y[i:n]$$

$$i = n$$

D([natura], D[nature])

D([natur], D[natur]) + [a] ↔ y[e]

D([natu], D[natu]) + [r] ↔ y[r]

Spelling Correction via Minimum Edit Distance

- Recursive Programing

$$D(x[0:n], y[0:n]) = D(x[0:i-1], y[0:i-1]) + \text{the cost of } x[i:n] \leftrightarrow y[i:n]$$

$$i = n$$

D([natura], D[nature])

D([natur], D[natur]) + [a] $\leftrightarrow$ y[e]

D([natu], D[natu]) + [r] $\leftrightarrow$ y[r]

D([nat], D[nat]) + [u] $\leftrightarrow$ y[u]

D([na], D[na]) + [t] $\leftrightarrow$ y[t]

D([n], D[n]) + [a] $\leftrightarrow$ y[a]

D([], D[]) + [n] $\leftrightarrow$ y[n]

D([], D[]) = 0

Spelling Correction via Minimum Edit Distance

- Recursive Programing

$$D(x[0:n], y[0:n]) = D(x[0:i-1], y[0:i-1]) + \text{the cost of } x[i:n] \leftrightarrow y[i:n]$$

$$i = n$$

D([natura], D[nature])

D([natur], D[natur]) + [a] ↔ y[e]

D([natu], D[natu]) + [r] ↔ y[r]

D([nat], D[nat]) + [u] ↔ y[u]

D([na], D[na]) + [t] ↔ y[t]

D([n], D[n]) + [a] ↔ y[a]

D([], D[]) + [n] ↔ y[n]

D([], D[]) = 0

Spelling Correction via Minimum Edit Distance

- Recursive Programing

$$D(x[0:n], y[0:n]) = D(x[0:i-1], y[0:i-1]) + \text{the cost of } x[i:n] \leftrightarrow y[i:n]$$
$$i = n$$

D([natura], D[nature])

D([natur], D[natur]) + [a] ↔ y[e]

D([natu], D[natu]) + [r] ↔ y[r]

D([nat], D[nat]) + [u] ↔ y[u]

D([na], D[na]) + [t] ↔ y[t]

D([n], D[n]) + [a] ↔ y[a]

0 + [n] ↔ y[n]

D([], D[]) = 0

Spelling Correction via Minimum Edit Distance

- Recursive Programing

$$D(x[0:n], y[0:n]) = D(x[0:i-1], y[0:i-1]) + \text{the cost of } x[i:n] \leftrightarrow y[i:n]$$
$$i = n$$

D([natura], D[nature])

D([natur], D[natur]) + [a] ↔ y[e]

D([natu], D[natu]) + [r] ↔ y[r]

D([nat], D[nat]) + [u] ↔ y[u]

D([na], D[na]) + [t] ↔ y[t]

D([n], D[n]) + [a] ↔ y[a]

0 + 0

D([], D[]) = 0

Spelling Correction via Minimum Edit Distance

- Recursive Programing

$$D(x[0:n], y[0:n]) = D(x[0:i-1], y[0:i-1]) + \text{the cost of } x[i:n] \leftrightarrow y[i:n]$$
$$i = n$$

D([natura], D[nature])

D([natur], D[natur]) + [a] ↔ y[e]

D([natu], D[natu]) + [r] ↔ y[r]

D([nat], D[nat]) + [u] ↔ y[u]

D([na], D[na]) + [t] ↔ y[t]

0 + [a] ↔ y[a]

0 + 0

D([], D[]) = 0

Spelling Correction via Minimum Edit Distance

- Recursive Programing

D(x[0:n], y[0:n]) = D(x[0:i-1], y[0:i-1]) + the cost of x[i:n] ↔ y[i:n]

i = n

D([natura], D[nature])

0 + [a] ↔ y[e]

0 + 0

0 + 0

0 + 0

0 + 0

0 + 0

D([], D[]) = 0

Spelling Correction via Minimum Edit Distance

- Recursive Programing

$$D(x[0:n], y[0:n]) = D(x[0:i-1], y[0:i-1]) + \text{the cost of } x[i:n] \leftrightarrow y[i:n]$$

$$i = n$$

D([natura], D[nature])

0 + 2

0 + 0

0 + 0

0 + 0

0 + 0

0 + 0

D([], D[]) = 0

Spelling Correction via Minimum Edit Distance

- Recursive Programing

$$D(x[0:n], y[0:n]) = D(x[0:i-1], y[0:i-1]) + \text{the cost of } x[i:n] \leftrightarrow y[i:n]$$

$$i = n$$

D([natura], D[nature]) = 2

Spelling Correction via Minimum Edit Distance

- Recursive Programing

Two words are similar if they are similar up until the last chars!

natural vs. naturl_
natur?l vs. naturl_

D([natural], [naturl_]) = D([natura], [naturl]) + [l] ↔ [_]
D([natur], [natur]) + [a] ↔ [l]

Spelling Correction via Minimum Edit Distance

- Recursive Programing

Two words are similar if they are similar up until the last chars!

natural vs. naturl_

D([natural], [naturl_]) = D([natura], [naturl]) + [l] ↔ [_]

0 + [a] ↔ [l]

Spelling Correction via Minimum Edit Distance

- Recursive Programing

Two words are similar if they are similar up until the last chars!

natural vs. naturl_

D([natural], [naturl_]) = 2 + [l] ↔ [_]

0 + 2

Spelling Correction via Minimum Edit Distance

- Recursive Programing

Two words are similar if they are similar up until the last chars!

natural vs. naturl_
natur?l vs. naturl_

D([natural], [naturl_]) = 2 + 1 = 3

Spelling Correction via Minimum Edit Distance

- Recursive Programing

Two words are similar if they are similar up until the last chars!

natural vs. naturl_

D([natural], [naturl_]) = D([natura], [natur]) + [l] ↔ [l]

D([natur], [natur]) + [a] ↔ []

Spelling Correction via Minimum Edit Distance

- Recursive Programing

Two words are similar if they are similar up until the last chars!

natural vs. naturl_

D([natural], [naturl_]) = D([natura], [natur]) + [l] ↔ [l]

0 + [a] ↔ []

Spelling Correction via Minimum Edit Distance

- Recursive Programing

Two words are similar if they are similar up until the last chars!

natural vs. naturl_

D([natural], [naturl_]) = 1 + [l] ↔ [l]

0 + 1

Spelling Correction via Minimum Edit Distance

- Recursive Programing

Two words are similar if they are similar up until the last chars!

natural vs. naturl_

D([natural], [naturl_]) = 1 + 0 = 1

Spelling Correction via Minimum Edit Distance

- Recursive Programing

$$D(x[0:n], y[0:n]) = D(x[0:i-1], y[0:i-1]) + \text{the cost of } x[i:n] \leftrightarrow y[i:n]$$

$$D(x[0:n], y[0:n]) = D(x[0:i], y[0:i-1]) + \text{the cost of } x[i+1:n] \leftrightarrow y[i:n]$$

Spelling Correction via Minimum Edit Distance

- Recursive Programing

$$D(x[0:n], y[0:n]) = D(x[0:i-1], y[0:i-1]) + \text{the cost of } x[i:n] \leftrightarrow y[i:n]$$

$$D(x[0:n], y[0:n]) = D(x[0:i], y[0:i-1]) + \text{the cost of } x[i+1:n] \leftrightarrow y[i:n]$$

$$D(x[0:n], y[0:n]) = D(x[0:i-1], y[0:i]) + \text{the cost of } x[i:n] \leftrightarrow y[i+1:n]$$

Spelling Correction via Minimum Edit Distance

- Recursive Programing

application	aplikatiion	Init
[ap]p[lication]	[ap][likatiion] → [ap]plikatiion	Insert (1)
[appli]c[ation]	[appli]k[atiion] → [appli]c[atiion]	Substitution (2)
[applicati][on]	[applicati]i[on] → [applicati][on]	Delete (1)
[application]	[application]	1+2+1 = 4

Spelling Correction via Minimum Edit Distance

- Recursive Programing

$$
D(x[0:n], y[0:n]) = \min\{
\begin{array}{l}
D(x[0:i-1], y[0:i-1]) + \text{the cost of } x[i:n] \leftrightarrow y[i:n] \\
D(x[0:i], y[0:i-1]) + \text{the cost of } x[i+1:n] \leftrightarrow y[i:n] \\
D(x[0:i-1], y[0:i]) + \text{the cost of } x[i:n] \leftrightarrow y[i+1:n]
\end{array}
\}
$$

Spelling Correction via Minimum Edit Distance

- Recursive Programing

$$D(x[0:n], y[0:n]) = \min\{$$

$$D(x[0:i-1], y[0:i-1]) + \text{the cost of } x[i:n] \leftrightarrow y[i:n]$$
$$D(x[0:i], y[0:i-1]) + \text{the cost of } x[i+1:n] \leftrightarrow y[i:n]$$
$$D(x[0:i-1], y[0:i]) + \text{the cost of } x[i:n] \leftrightarrow y[i+1:n]$$

$$\}$$

Does not work! (why?)

Spelling Correction via Minimum Edit Distance

Levenshtein (1966)

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

  n ← LENGTH(source)
  m ← LENGTH(target)
  Create a distance matrix distance[n+1,m+1]

  # Initialization: the zeroth row and column is the distance from the empty string
      D[0,0] = 0
      for each row i from 1 to n do
          D[i,0] ← D[i-1,0] + del-cost(source[i])
      for each column j from 1 to m do
          D[0,j] ← D[0, j-1] + ins-cost(target[j])

  # Recurrence relation:
  for each row i from 1 to n do
      for each column j from 1 to m do
          D[i, j] ← MIN( D[i−1, j] + del-cost(source[i]),
                         D[i−1, j−1] + sub-cost(source[i], target[j]),
                         D[i, j−1] + ins-cost(target[j]))
  # Termination
  return D[n,m]
```

Spelling Correction via Minimum Edit Distance

i

j

	a	p	p	l	i	c	a	t	i	o	n
a											
p											
l											
i											
k											
a											
t											
i											
i											
n											

$$
D(x[0{:}i], y[0{:}j]) = \min\{
\begin{array}{l}
D(x[0{:}i-1], y[0{:}j-1]) + x[i] \leftrightarrow y[j] \\
D(x[0{:}i], y[0{:}j-1]) + 1 \\
D(x[0{:}i-1], y[0{:}j]) + 1
\end{array}
\}
$$

$x[i] \neq y[j] \rightarrow 2$

$x[i] = y[j] \rightarrow 0$

Spelling Correction via Minimum Edit Distance

i

j

	a	p	p	l	i	c	a	t	i	o	n
a											
p											
l											
i											
k											
a											
t											
i											
i											
n											

$$
\begin{aligned}
D(x[0:i], y[0:j]) = \min\{ \quad & \\
& D(x[0:i-1], y[0:j-1]) + x[i] \leftrightarrow y[j] \\
& D(x[0:i], y[0:j-1]) + 1 \\
& D(x[0:i-1], y[0:j]) + 1 \\
& \}
\end{aligned}
$$

$D(x[0:n], y[0:m])$

$x[i] \neq y[j] \rightarrow 2$

$x[i] = y[j] \rightarrow 0$

Spelling Correction via Minimum Edit Distance

	a	p	p	l	i	c	a	t	i	o	n
a	0	1	2	3	4	5	6	7	8	9	10
p	1										
l	2										
i	3										
k	4										
a	5										
t	6										
i	7										
i	8										
n	9										

D(x[0:n], y[0:m])

Spelling Correction via Minimum Edit Distance

	a	p	p	l	i	c	a	t	i	o	n
a	0	1	2	3	4	5	6	7	8	9	10
p	1	0	1	2	3	4	5	6	7	8	9
l	2										
i	3										
k	4										
a	5										
t	6										
i	7										
i	8										
n	9										

D(x[0:n], y[0:m])

Backtrace:

1. No Change
2. No Change
3. Insert
4. Insert
5. Insert
6. Insert
7. ...

Spelling Correction via Minimum Edit Distance

	a	p	p	l	i	c	a	t	i	o	n
a	0	1	2	3	4	5	6	7	8	9	10
p	1	0	1	2	3	4	5	6	7	8	9
l	2	1									
i	3	2									
k	4	3									
a	5	4									
t	6	5									
i	7	6									
i	8	7									
n	9	8									

D(x[0:n], y[0:m])

Backtrace:
1. No Change
2. No Change
3. Insert
4. Insert
5. Insert
6. Insert
7. ...

Spelling Correction via Minimum Edit Distance

	a	p	p	l	i	c	a	t	i	o	n
a	0	1	2	3	4	5	6	7	8	9	10
p	1	0	1	2	3	4	5	6	7	8	9
l	2	1	2	1							
i	3	2	3	2							
k	4	3									
a	5	4									
t	6	5									
i	7	6									
i	8	7									
n	9	8									

D(x[0:n], y[0:m])

Backtrace:
1. No Change
2. No Change
3. Insert
4. Delete

Spelling Correction via Minimum Edit Distance

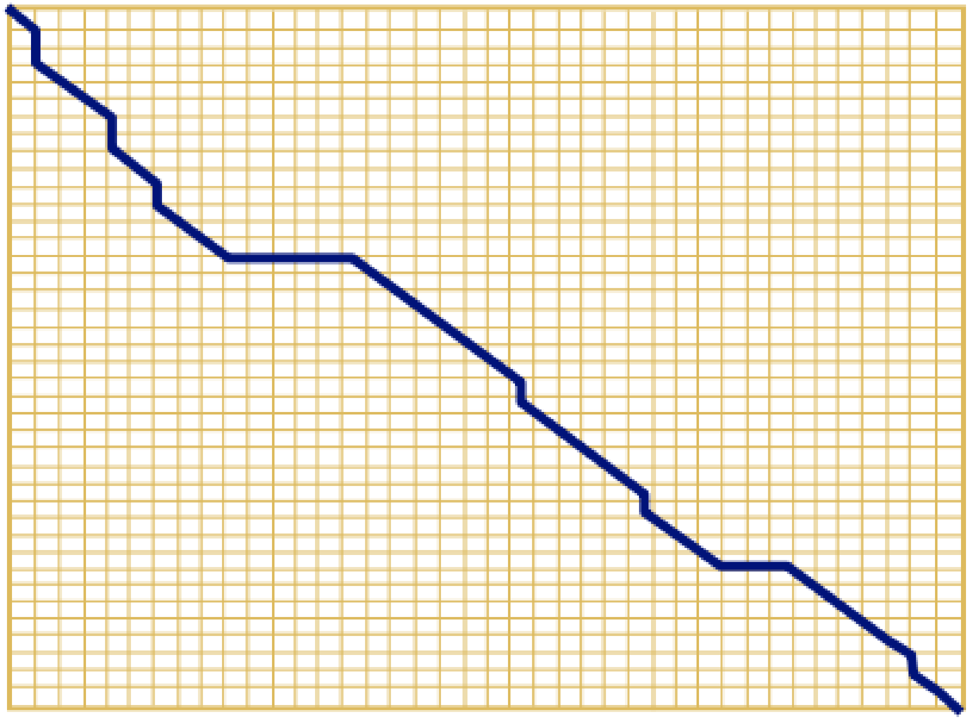
```python

levenshtein("application", "aplikatiion")

```

```
00 01 02 03 04 05 06 07 08 09 10 11
01 00 01 02 03 04 05 06 07 08 09 10
02 01 00 01 02 03 04 05 06 07 08 09
03 02 01 02 03 04 05 06 07 08 09 10
04 03 02 01 02 03 04 05 06 07 08 09
05 04 03 02 01 02 03 04 05 06 07 08
06 05 04 03 02 03 04 05 06 07 08 09
07 06 05 04 03 04 03 04 05 06 07 08
08 07 06 05 04 05 04 03 04 05 06 07
09 08 07 06 05 06 05 04 03 04 05 06
10 09 08 07 06 07 06 05 04 05 04 05
11 10 09 08 07 08 07 06 05 06 05 04
4.0
```

Spelling Correction via Minimum Edit Distance



from (0,0) to (M, N)

corresponds to
an alignment
of the two sequences

An optimal alignment is composed
of optimal subalignments

Dan Jurafsky

Spelling Correction via Minimum Edit Distance

Levenshtein (1966): Complexity

- Time: ?
- Space: ?

Spelling Correction via Minimum Edit Distance

Levenshtein (1966): Complexity: Filling the matrix

- Time: O(n*m)
- Space: O(n*m)

Learn to Spelling Correction

Keyboard

Online texts (e.g., emails) depends on keyboards.

1. Misspells happens more on characters that sit next to each other on the keyboard.
2. Speed of typing is a source of error → Transposition: ‘Desing’ ‘Design’

Weighted Minimum 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

Spelling Correction via Minimum Edit Distance

$$D(x[0{:}i], y[0{:}j]) = \min\{$$

$$D(x[0{:}i\text{-}1], y[0{:}j\text{-}1]) + x[i] \leftrightarrow y[j]$$

$$D(x[0{:}i], y[0{:}j\text{-}1]) + \text{insert}(y[j])$$

$$D(x[0{:}i\text{-}1], y[0{:}j]) + \text{delete}(x[i])$$

$$\}$$

$x[i] \neq y[j] \rightarrow \text{sub}(x[i], y[j])$

$x[i] = y[j] \rightarrow 0$

Spelling Correction via Minimum Edit Distance

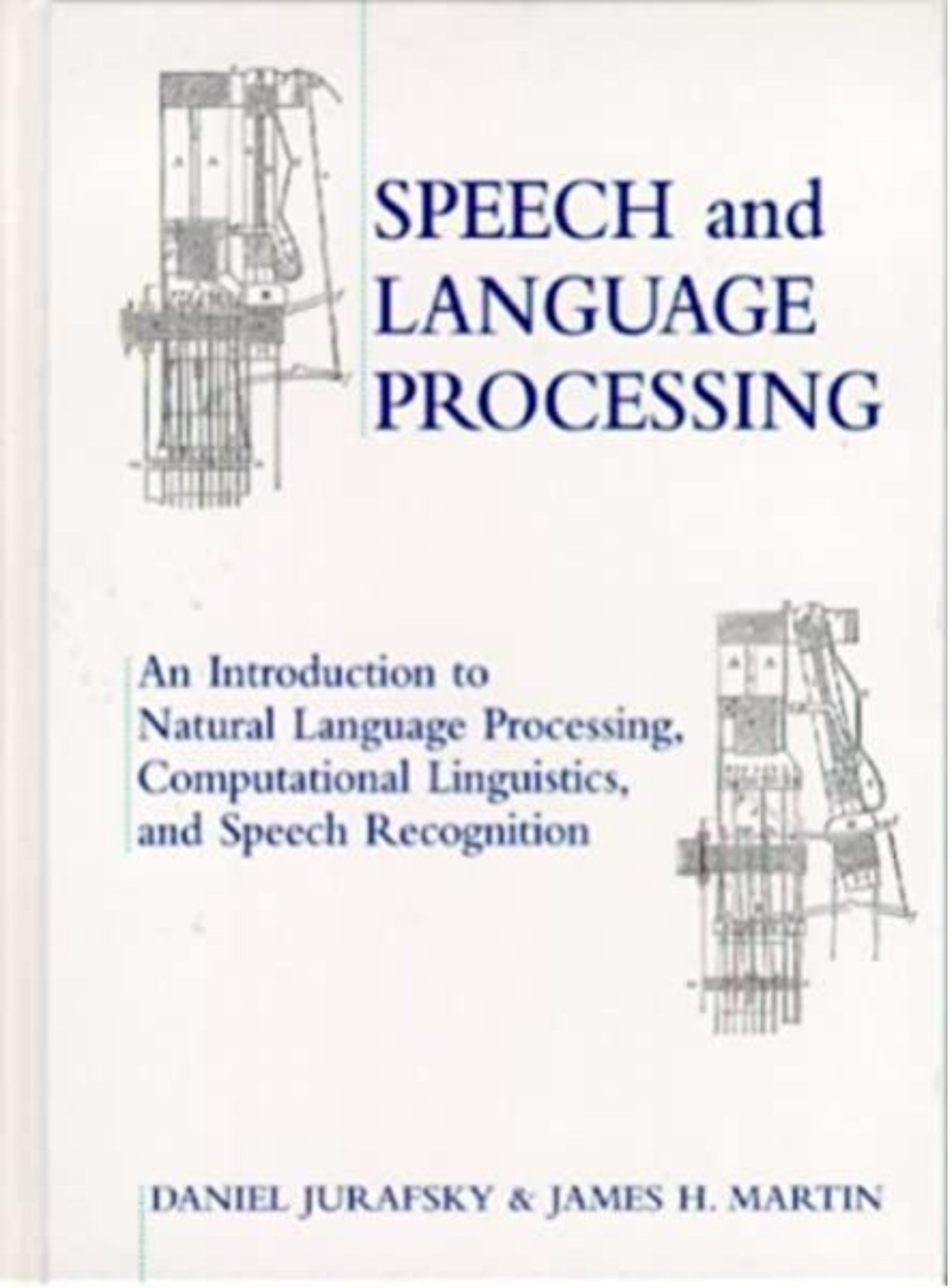
Applications:

1. Finding the closest word from Dictionary as the correct spell
   Autocorrection

2. Finding the closest word from Dictionary as the correct meaning!?

3. Finding the closest word from Dictionary as the prediction!?
   Autocompletion

4. Computational Biology
   Aligning two sequences of protein
   Daniel Jurafsky: https://www.youtube.com/watch?v=IL0-bD_e8s4


SPEECH and
LANGUAGE
PROCESSING
An Introduction to
Natural Language Processing,
Computational Linguistics,
and Speech Recognition
DANIEL JURAFSKY & JAMES H. MARTIN