CS 481

Introduction to NLP

February 2, 2023

Announcements / Reminders

- Please follow the Week 04 To Do List instructions
- Quiz #03 due on Sunday (02/05/23) at 11:59 PM CST
- WA #01 due TODAY (02/02/23) at 11:59 PM CST
- PA #01 due on Monday (02/20/23) at 11:59 PM CST

- Exam dates:
  - Midterm: 03/02/2023 during Thursday lecture time
  - Final: 04/27/2023 during Thursday lecture time

Teaching Assistant

Name	e-mail
Lan Wei	lwei3@hawk.iit.edu

TAs will:

- grade your assignments

Add a **[CS481 Spring 2023]** prefix to your email subject when contacting TAs, please.

Plan for Today

- **N-grams and language models**
- **Spelling: Minimum Edit Distance**
- **Parts of Speech tagging – introduction (if time permits)**

What is an N-Gram

An N-gram is a subsequence of N items from a given sequence.

- **unigram: n-gram of size 1**
- **bigram (or Digram): n-gram of size 2**
- **trigram: n-gram of size 3**

Item:

- **phonemes**
- **syllables**
- **letters**
- **words**
- **anything else depending on the application.**

The Idea

- **Examine short sequences of words**
- **How likely is each sequence?**
- **Use Markov assumption / property**

Chain Rule

Conditional probabilities can be used to decompose conjunctions using the chain rule. For random variables $\boldsymbol{f_1, f_2, \ldots, f_n}$:

$P(f_1 \wedge f_2 \wedge \ldots \wedge f_n) =$

$P(f_1)\ *$

$P(f_2 \mid f_1)\ *$

$P(f_3 \mid f_1 \wedge f_2)\ *$

$\ldots$

$P(f_n \mid f_1 \wedge \ldots \wedge f_{n-1}) =$

$= \prod_{i=1}^{n} P(f_i \mid Parents(f_i))$ $\leftarrow$ Enabled by conditional independence

The Idea: Markov Assumption

- **The probability of the appearance of a word depends on the words that have appeared before it.**

$$P(\text{rabbit} \mid \text{Just then the white})$$

- **Impossible to calculate this probability from a corpus. The exact word sequence would have to appear in the corpus.**
- **Markov simplifying assumption: we approximate the probability of a word given all the previous words with the probability given only the previous word.**

$$P(\text{rabbit} \mid \text{Just then the white}) \approx P(\text{rabbit} \mid \text{white})$$

Probabilistic Language Models

- **Task A: compute the joint probability of a sentence or sequence of words**

$$P(W) = P(w_1, w_2, w_3, w_4, w_5, \ldots, w_n)$$

- **Task B: compute conditional probability of an upcoming word**

$$P(w_5 \mid w_1, w_2, w_3, w_4) = ?$$

- **A model that can compute either**

$$P(W) \quad \textbf{or} \quad P(w_n \mid w_1, w_2, \ldots, w_{n-1})$$

is called a language model (LM).

$w_1, w_2, w_3, w_4, w_5, \ldots, w_n$ - words

Probabilities and Their Estimates

- **Probability of a single word (token) occuring**

$$P(word) \approx \frac{count(word)}{count(all\ words\ /\ tokens)}$$

- **Probability of a sequence of words (tokens) occurring (where w_i - ith word / token)**

By chain rule:

$$P(w_1 = x_1 \wedge w_2 = x_2 \wedge \ldots \wedge w_n = xn) = \prod_{i=1}^{n} P(w_i = xi \mid w_1 = x_1 \wedge \ldots \wedge w_{i-1} = xi\_1)$$

$$P(first, second, .., \quad nth) = \prod_{i=1}^{n} P(ith \mid all\ words\ preceding\ ith)$$

By Markov assumption:

$$P(next \mid all\ words\ preceding\ next) \approx P(next \mid N - gram\ preceding\ next)$$

Probabilities and Their Estimates

Probability of a sequence of words (tokens), an N-gram ($[last\ K\ words\ before\ next], next$), occurring:

$$P(next \mid last\ K\ words\ before\ next) =$$

$$= \frac{count([last\ K\ words\ before\ next]\ next)}{\sum_{x \in V} count([last\ K\ words\ before\ next]\ x)} =$$

$$= \frac{count([last\ K\ words\ preceding\ next]\ next)}{count([last\ K\ words\ preceding\ next])}$$

Probabilities and Their Estimates

Probability of a sequence of words (tokens), an N-gram ($[\boldsymbol{prefix}], \boldsymbol{next}$), occurring:

$$\boldsymbol{P}(\boldsymbol{next} \mid [\boldsymbol{prefix}]\ \boldsymbol{next}) = \frac{\boldsymbol{count}([\boldsymbol{prefix}]\ \boldsymbol{next})}{\sum_{\boldsymbol{x} \in \boldsymbol{V}} \boldsymbol{count}([\boldsymbol{prefix}]\ \boldsymbol{x})} =$$

$$= \frac{\boldsymbol{count}([\boldsymbol{prefix}]\ \boldsymbol{next})}{\boldsymbol{count}(\boldsymbol{prefix})} =$$

$$= \frac{\boldsymbol{observed\ frequency\ of\ a\ }([\boldsymbol{prefix}] + \boldsymbol{next})\ \boldsymbol{sentence}}{\boldsymbol{observed\ frequency\ of\ a\ prefix}} =$$

$$= \boldsymbol{relative\ frequency\ of}([\boldsymbol{prefix}] + \boldsymbol{next})$$

Unigram Language Model

Zeroth-order Markov Assumption:

$$P(w_1, w_2, \ldots, w_n) \approx \prod_{i=1}^{n} P(w_i)$$

Bigram Language Model

First-order Markov Assumption:

$$P(w_1, w_2, \ldots, w_{i-1}) \approx P(w_i \mid w_{i-1})$$

General Maximum Likelihood Estimation (MLE) of an 2-gram (bigram):

$$P(w_N \mid w_{N-1}) = \frac{count(w_{N-1}, w_N)}{count(w_{N-1})}$$

where:

w_i - **ith word / token**

Trigram Language Model

Second-order Markov Assumption:

$$P(w_i \mid w_1, w_2, \ldots, w_{i-1}) \approx P(w_i \mid w_{i-2}, w_{i-1})$$

General Maximum Likelihood Estimation (MLE) of an 2-gram (bigram):

$$P(w_N \mid w_{N-2}, w_{N-1}) = \frac{count(w_{N-2}, w_{N-1}, w_N)}{count(w_{N-2}, w_{N-1})}$$

where:

w_i **- ith word / token**

N-gram Language Models

General Maximum Likelihood Estimation (MLE) of an N-gram:

$$P(w_N \mid w_{N-K+1}, w_{N-K+2}, \ldots, w_{N-1}) = \frac{count(w_{N-K+1}, w_{N-K+2}, \ldots, w_{N-1}, w_N)}{count(w_{N-K+1}, w_{N-K+2}, \ldots, w_{N-1})}$$

where:

w_i - **ith word / token**

In MLE, the resulting parameter set maximizes the likelihood of the training set T given the model M (i.e., P(T | M)).

N-gram Language Models

- **We can extend to 4-grams, 5-grams**

  In general this is an insufficient model of language, because language has long-distance dependencies:

  "The computer(s) which I had just put into the machine room on the fifth floor is (are) crashing."

- **But in many cases N-gram models suffice**

Example: A Simple Corpus

Consider the following corpus (three sentences):

I am Sam

Sam I am

I do not like green eggs and ham

Example: A Simple Corpus

Let’s add sentence start / end tokens:

<s> *I am Sam* </s>

<s> *Sam I am* </s>

<s> *I do not like green eggs and ham* </s>

This will help us create “starts with” and “ends with” bigrams.

Example: A Simple Corpus

Let’s add sentence start / end tokens:

<s> *I am Sam* </s>

<s> *Sam I am* </s>

<s> *I do not like green eggs and ham* </s>

Let’s list all unique **unigrams** first:

I, am, Sam, do, not, like, green, eggs, and, ham

And let’s count their occurences:

	I	am	Sam	do	not	like	green	eggs	and	ham
c(word)	3	2	2	1	1	1	1	1	1	1

Example: A Simple Corpus

Given our corpus:

<s> *I am Sam* </s>

<s> *Sam I am* </s>

<s> *I do not like green eggs and ham* </s>

and word token frequency counts:

	I	am	Sam	do	not	like	green	eggs	and	ham
c(word)	3	2	2	1	1	1	1	1	1	1

Bigram probability estimate:

$$P(Sam \mid am) = P(w_N \mid w_{N-1}) = \frac{count(w_{N-1}, w_N)}{count(w_{N-1})} = \frac{count(am, Sam)}{count(am)} = \frac{1}{2}$$

Example: A Simple Corpus

Given our corpus:

<s> *I am* *Sam* </s>

<s> *Sam* *I am* </s>

<s> *I do not like green eggs and ham* </s>

and word token frequency counts:

	I	am	Sam	do	not	like	green	eggs	and	ham
c(word)	3	2	2	1	1	1	1	1	1	1

Bigram probability estimate:

$$P(am \mid I) = P(w_N \mid w_{N-1}) = \frac{count(w_{N-1}, w_N)}{count(w_{N-1})} = \frac{count(I, am)}{count(I)} = \frac{2}{3}$$

Example: A Simple Corpus

Given our corpus:

<s> *I am Sam* </s>

<s> *Sam I am* </s>

<s> *I do not like green eggs and ham* </s>

and word token frequency counts:

	I	am	Sam	do	not	like	green	eggs	and	ham
c(word)	3	2	2	1	1	1	1	1	1	1

Bigram probability estimate:

$$P(do \mid I) = P(w_N \mid w_{N-1}) = \frac{count(w_{N-1}, w_N)}{count(w_{N-1})} = \frac{count(I, do)}{count(I)} = \frac{1}{3}$$

Example: A Simple Corpus

Given our corpus:

<s> I am Sam </s>

<s> Sam I am </s>

<s> I do not like green eggs and ham </s>

and word token frequency counts:

	I	am	Sam	do	not	like	green	eggs	and	ham
c(word)	3	2	2	1	1	1	1	1	1	1

Bigram probability estimate:

$$P(I \mid <s>) = P(w_N \mid w_{N-1}) = \frac{count(w_{N-1}, w_N)}{count(w_{N-1})} = \frac{count(<s>, I)}{count(<s>)} = \frac{2}{3}$$

Example: A Simple Corpus

Given our corpus:

</s> *I am Sam </s>*

</s> *Sam I am </s>*

</s> *I do not like green eggs and ham </s>*

and word token frequency counts:

	I	am	Sam	do	not	like	green	eggs	and	ham
c(word)	3	2	**2**	1	1	1	1	1	1	1

Bigram probability estimate:

$$P(</s> \mid Sam) = P(w_N \mid w_{N-1}) = \frac{count(I, do)}{count(I)} = \frac{count(Sam, </s>)}{count(Sam)} = \frac{1}{2}$$

Example: A More Complex Corpus

The Berkeley Restaurant Project (BeRP) was a testbed for a speech recognition system developed by the International Computer Science Institute (ICSI) in Berkeley, CA, USA, in the 1990's.

The BeRP system was designed to be an automated consultant whose domain of knowledge was restaurants in the city of Berkeley. The system served as a testbed for several research projects, including robust feature extraction, neural-net based phonetic likelihood estimation, automatic induction of multiple pronunciation lexicons, foreign accent detection and modeling, advanced language models, and lip-reading.

Example: A More Complex Corpus

Selected sentences from BeRP:

can you tell me about any good cantonese restaurants close by
mid priced thai food is what i'm looking for
tell me about chez panisse
can you give me a listing of the kinds of food that are available
i'm looking for a good place to eat breakfast
when is caffe venezia open during the day

Example: A More Complex Corpus

Selected sentences from BeRP (9332 sentences, V = 1446 words):

<s> can you tell me about any good cantonese restaurants close by </s>
<s> mid priced thai food is what i'm looking for </s>
<s> tell me about chez panisse </s>
<s> can you give me a listing of the kinds of food that are available </s>
<s> i'm looking for a good place to eat breakfast </s>
<s> when is caffe venezia open during the day </s>

with "sentence start" and "sentence end" tokens.

BeRP Selected Bigrams: Counts

BeRP bigram **counts** for selected words/tokens:

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

Recall the formula for MLE of a bigram:

$$P(w_N \mid w_{N-1}) = \frac{count(w_{N-1}, w_N)}{count(w_{N-1})} \leftarrow \boxed{827} = \text{count}(\textit{i}, \textit{want})$$

BeRP Selected Bigrams: Counts

BeRP bigram **counts** for selected words/tokens:

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	$\boxed{2}$	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

Recall the formula for MLE of a bigram:

$$P(w_N \mid w_{N-1}) = \frac{count(w_{N-1}, w_N)}{count(w_{N-1})} \leftarrow \boxed{2} = \text{count}(\textit{want}, \textit{i})$$

BeRP Selected Bigrams: Counts

BeRP bigram counts for selected words/tokens:

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

Recall the formula for MLE of a bigram:

$$P(w_N \mid w_{N-1}) = \frac{count(w_{N-1}, w_N)}{count(w_{N-1})}$$

$\boxed{5}$ = count(*i*, *i*)

Corpus Bigram Counts: Normalizing

Let's try to turn counts into probabilities:

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

Unigram counts (within the corpus) for all the words above:

	i	want	to	eat	chinese	food	lunch	spend
c(word)	2533	927	2417	746	158	1093	341	278

Corpus Bigram Counts: Normalizing

Let's try to turn counts into probabilities:

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

Recall the formula for MLE of a bigram:

$$P(w_N \mid w_{N-1}) = \frac{count(w_{N-1}, w_N)}{count(w_{N-1})}$$

Corpus Bigram Counts: Normalizing

Normalizing (using (N-1)-gram counts):

	i	want	to	eat	chinese	food	lunch	spend
i	5/2533	827/2533	0/2533	9/2533	0/2533	0/2533	0/2533	2/2533
want	2/927	0/927	608/927	1/927	6/927	6/927	5/927	1/927
to	2/2417	0/2417	4/2417	686/2417	2/2417	0/2417	6/2417	211/2417
eat	0/746	0/746	2/746	0/746	16/746	2/746	42/746	0/746
chinese	1/153	0/153	0/153	0/153	0/153	82/153	1/153	0/153
food	15/1093	0/1093	15/1093	0/1093	1/1093	4/1093	0/1093	0/1093
lunch	2/341	0/341	0/341	0/341	0/341	1/341	0/341	0/341
spend	1/278	0/278	1/278	0/278	0/278	0/278	0/278	0/278

Unigram counts (within the corpus) for all the words above:

	i	want	to	eat	chinese	food	lunch	spend
c(word)	2533	927	2417	746	158	1093	341	278

Corpus Bigram Counts: Normalizing

Normalizing:

	i	want	to	eat	chinese	food	lunch	spend
i	5/2533	827/2533	0/2533	9/2533	0/2533	0/2533	0/2533	2/2533
want	2/927	0/927	608/927	1/927	6/927	6/927	5/927	1/927
to	2/2417	0/2417	4/2417	686/2417	2/2417	0/2417	6/2417	211/2417
eat	0/746	0/746	2/746	0/746	16/746	2/746	42/746	0/746
chinese	1/153	0/153	0/153	0/153	0/153	82/153	1/153	0/153
food	15/1093	0/1093	15/1093	0/1093	1/1093	4/1093	0/1093	0/1093
lunch	2/341	0/341	0/341	0/341	0/341	1/341	0/341	0/341
spend	1/278	0/278	1/278	0/278	0/278	0/278	0/278	0/278

Unigram counts:

	i	want
c(word)	**2533**	927

$$P(w_N \mid w_{N-1}) = \frac{count(w_{N-1}, w_N)}{count(w_{N-1})}$$

5 / 2533

Corpus Bigram Counts: Normalizing

Normalizing:

	i	want	to	eat	chinese	food	lunch	spend
i	5/2533	827/2533	0/2533	9/2533	0/2533	0/2533	0/2533	2/2533
want	2/927	0/927	608/927	1/927	6/927	6/927	5/927	1/927
to	2/2417	0/2417	4/2417	686/2417	2/2417	0/2417	6/2417	211/2417
eat	0/746	0/746	2/746	0/746	16/746	2/746	42/746	0/746
chinese	1/153	0/153	0/153	0/153	0/153	82/153	1/153	0/153
food	15/1093	0/1093	15/1093	0/1093	1/1093	4/1093	0/1093	0/1093
lunch	2/341	0/341	0/341	0/341	0/341	1/341	0/341	0/341
spend	1/278	0/278	1/278	0/278	0/278	0/278	0/278	0/278

Unigram counts:

	i	want
c(word)	2533	**927**

$$P(w_N \mid w_{N-1}) = \frac{count(w_{N-1}, w_N)}{count(w_{N-1})}$$

2 / 927

Bigram Probability Estimates

Bigram probability estimates after normalizing:

	i	want	to	eat	chinese	food	lunch	spend
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.0014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

Bigram Probability Estimates

Bigram probability estimates after normalizing:

	i	want	to	eat	chinese	food	lunch	spend
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.0014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

Let's introduce a couple more probabilities:

P(*i* | *<s>*) = 0.25 **P(*english* | *want*) = 0.0011**

P(*food* | *english*) = 0.5 **P(*</s>* | *food*) = 0.68**

Bigram Probability Estimates

Bigram probability estimates after normalizing:

	i	want	to	eat	chinese	food	lunch	spend
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.0014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

Now we are ready to calculate probability of some sentence:

$$P(first, second, .., \quad nth) = \prod_{i=1}^{n} P(ith \mid all\ words\ preceding\ ith)$$

Bigram Probability Estimates

Bigram probability estimates after normalizing:

	i	want	to	eat	chinese	food	lunch	spend
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.0014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

Now we are ready to calculate probability of some sentence *S1* (“*I want English food*”):

$$\boldsymbol{P}(\boldsymbol{S1}) = \boldsymbol{P}(<\mathrm{s}>, \mathrm{I}, \mathrm{want}, \mathrm{english}, \mathrm{food}, </\mathrm{s}>) = \prod_{i=1}^{n} \boldsymbol{P}(\boldsymbol{ith} \mid \boldsymbol{all\ words\ preceding\ ith})$$

Bigram Probability Estimates

Bigram probability estimates after normalizing:

	i	want	to	eat	chinese	food	lunch	spend
i	0.002	**0.33**	0	0.0036	0	0	0	0.00079
want	0.022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.0014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

Now we are ready to calculate probability of some sentence *S1* ("*I want English food*"):

$$\boldsymbol{P(S1) = P(i \mid <s>) * P(want \mid i) * P(english \mid want) * P(food \mid english) * P(</s> \mid food)}$$
$$\boldsymbol{= 0.25 * 0.33 * 0.0011 * 0.5 * 0.68 \approx 0.000031}$$

Bigram Probability Estimates

Bigram probability estimates after normalizing:

	i	want	to	eat	chinese	food	lunch	spend
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.0014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

Let's calculate probability of another sentence *S2* ("*I want Chinese food*"):

$$P(S2) = P(<\mathrm{s}>, \mathrm{I}, \mathrm{want}, \mathbf{chinese}, \mathrm{food}, </\mathrm{s}>) = \prod_{i=1}^{n} P(ith \mid all\ words\ preceding\ ith)$$

Bigram Probability Estimates

Bigram probability estimates after normalizing:

	i	want	to	eat	chinese	food	lunch	spend
i	0.002	**0.33**	0	0.0036	0	0	0	0.00079
want	0.022	0	0.66	0.0011	**0.0065**	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	**0.52**	0.0063	0
food	0.014	0	0.0014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

Let's calculate probability of another sentence *S2* ("*I want Chinese food*"):

$$\boldsymbol{P(S2) = P(i \mid <s>) * P(want \mid i) * P(chinese \mid want) * P(food \mid chinese) * P(</s> \mid food)}$$
$$\boldsymbol{= 0.25 * 0.33 * 0.0065 * 0.52 * 0.68 \approx 0.00019}$$

Bigram Probability Estimates

Bigram probability estimates after normalizing:

	i	want	to	eat	chinese	food	lunch	spend
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.0014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

Which sentence is more likely: "*I want English food***" or "***I want Chinese food***"?**

$$\boldsymbol{P(S1) = P(I\ want\ English\ food)\ \approx 0.000031}$$
$$\boldsymbol{P(S2) = P(I\ want\ Chinese\ food)\ \approx 0.00019}$$

In Practice: Calculations

- **Perform calculations in log space**

log(P1 * P2 * P3 * P4) = log P1 + log P2 + log P3 + log P4

  - **adding faster than multiplying**
  - **avoids underflow**

How Good Is Your Model?

- Does our language model prefer good sentences to bad ones?
  - Assigns higher probability to "real" or "frequently observed" sentences (as opposed to "ungrammatical" or "rarely observed" sentences)?
- We train parameters of our model on a training set
- We test the model's performance on data we haven't seen:
  - a test set: unseen dataset that is different from training set
  - an evaluation metric tells us how well our model does on the test set.

Extrinsic Evaluation of N-gram Models

- Best evaluation for comparing models A and B
  - apply each model to a specific task (spelling corrector, speech recognizer, machine translation)
  - Run the task, get accuracy for both A and B
    - How many misspelled words corrected properly?
    - How many words translated correctly?
  - Compare accuracy for A and B

Extrinsic Evaluation of N-gram Models

- Extrinsic evaluation
  - Time-consuming; can take days or weeks
  - Bad approximation
    - unless the test data looks just like the training data
    - generally only useful in pilot experiments
  - But is helpful to do

- Alternatives:
  - use intrinsic evaluation: perplexity

Intrinsic Evaluation: Perplexity

- **The best language model is the one that best predicts an unseen test set**
  - **Gives the highest P(*sentence*)**
- **Perplexity is the inverse probability of the test set, normalized by the number of words N:**

$$PP(W) = P(w_1 w_2 \ldots w_N)^{-\frac{1}{N}}$$

$$= \sqrt[N]{\frac{1}{P(w_1 w_2 \ldots w_N)}}$$

by Chain Rule: $$\mathrm{PP}(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_1 \ldots w_{i-1})}}$$

for bigrams: $$\mathrm{PP}(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_{i-1})}}$$

Minimizing perplexity is the same as maximizing probability

In Practice: Sparse Matrix

BeRP bigram **counts** for selected words/tokens:

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

Note: This is a sparse matrix (lots of zeros)!

In Practice: Zeros

Training set:

… denied the allegations
… denied the reports
… denied the claims
… denied the request

Test set:

… denied the offer
… denied the loan

P(*offer* | *denied the*) = 0

In Practice: Smoothing

- Smoothing removes zero-probabilities
- Smoothing assigns probabilities to unseen events

- There are many smoothing algorithms:
  - simple: Laplace / Add One
  - “stupid backoff”
  - advanced: Extended Interpolated Kneser-Nay

Laplace / Add One Smoothing

Pretend you saw everything +1 times:

	i	want	to	eat	chinese	food	lunch	spend
i	5+1	827+1	0+1	9+1	0+1	0+1	0+1	2+1
want	2+1	0+1	608+1	1+1	6+1	6+1	5+1	1+1
to	2+1	0+1	4+1	686+1	2+1	0+1	6+1	211+1
eat	0+1	0+1	2+1	0+1	16+1	2+1	42+1	0+1
chinese	1+1	0+1	0+1	0+1	0+1	82+1	1+1	0+1
food	15+1	0+1	15+1	0+1	1+1	4+1	0+1	0+1
lunch	2+1	0+1	0+1	0+1	0+1	1+1	0+1	0+1
spend	1+1	0+1	1+1	0+1	0+1	0+1	0+1	0+1

Updated unigram probability:

$$P_{addOne}(w_N) = \frac{count(w_N) + 1}{count(all\ words/tokens) + number\ of\ unique\ words/tokens\ V}$$

In Practice: Out-Of-Vocabulary Words

- If we know all the words in advance, so the vocabulary V is fixed
  - closed vocabulary task
- In practice often we don't know the entire vocabulary
  - Out Of Vocabulary = OOV words
  - open vocabulary task

- Potential solution: create an unknown word token $<\mathrm{UNK}>$
  - Training of <UNK> probabilities
    - Create a fixed lexicon L of size V
    - At text normalization phase, any training word not in L changed to $<\mathrm{UNK}>$
    - Now we train its probabilities like a normal word
  - At decoding time
    - If text input: Use $<\mathrm{UNK}>$ probabilities for any word not in training

Spelling: Real-world Problems

- **Non-word error detection**
  - *graffe* **instead of** giraffe
- **Isolated-word error correction**
- **Context-dependent error detection and correction**
  - **typos**
    - *three* **instead of** *there*
  - **homophone or near-homophones**
    - *dessert* **instead of** *desert* **or** *piece* **for** *peace*

How Similar are Two Strings?

- **The user typed "*graffe*". Which string is closest?**
  - *graf*
  - *graft*
  - *grail*
  - *giraffe*

- **Why? Spell checking**

How Similar are Two Strings?

- **Why? Computational Biology:**
  - **Align two sequences of nucleotides:**

```
AGGCTATCACCTGACCTCCAGGCCGATGCCC
TAGCTATCACGACCGCGGTCGATTTGCCCGAC
```

  - **Resulting alignment:**

```
-AGGCTATCACCTGACCTCCAGGCCGA--TGCCC---
TAG-CTATCAC--GACCGC--GGTCGATTTGCCCGAC
```

How Similar are Two Strings?

- **The user typed "*graffe*". Which string is closest?**
  - *graf* **deleted** "*i*" **deleted** "*fe*"
  - *graft* **deleted** "*i*" "*e*" **and substituted** "*f*"
  - *grail* **deletion and substitution**
  - *giraffe* **correct form (we need to insert** "*i*")

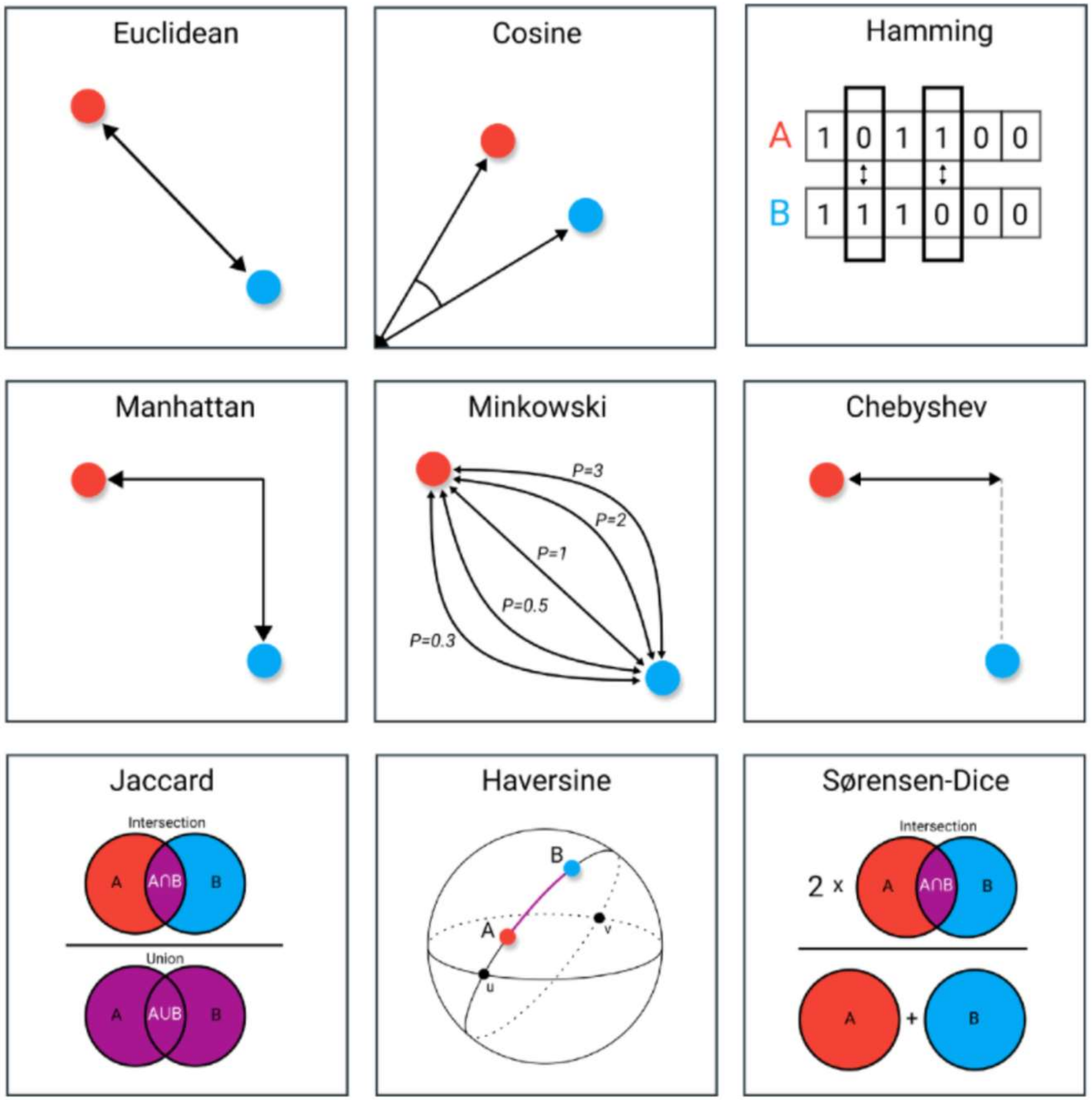
Alignment

Given two sequences, an alignment is a correspondence between substrings of the two sequences.

```
I N T E * N T I O N
| | | | | | | | | |
* E X E C U T I O N
```

Alignment is made up of edits.

Distance Measures | String Distance?



Source: https://towardsdatascience.com/9-distance-measures-in-data-science-918109d069fa

Edits

One string can be transformed to another by a sequence of edits (delete**, i**nsert**, s**ubstitute**).**

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

Edits with Costs: Edit Distance

Each edit operation can have its cost:

- **cost(d) = cost(i) = cost(s) = 1**

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

Edit distance = 5

Edits with Costs: Levenshtein Distance

Each edit operation can have its cost:

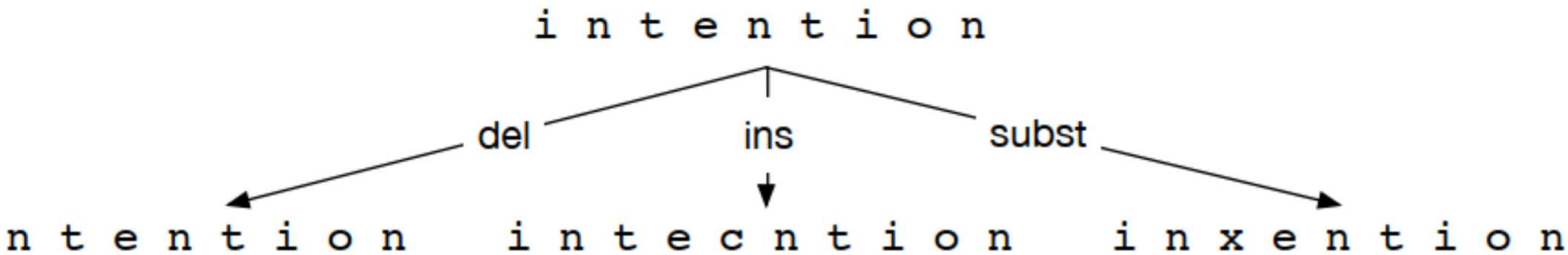
- **cost(d) = cost(i) = 1 | cost(s) = cost(d) + cost(i) = 2**

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

Levenshtein edit distance = 8

Searching for Minimum Edit Path

String transformation (a sequence of edits) can be represented with a tree:



Solution: Minimum Edit Path found via tree search:

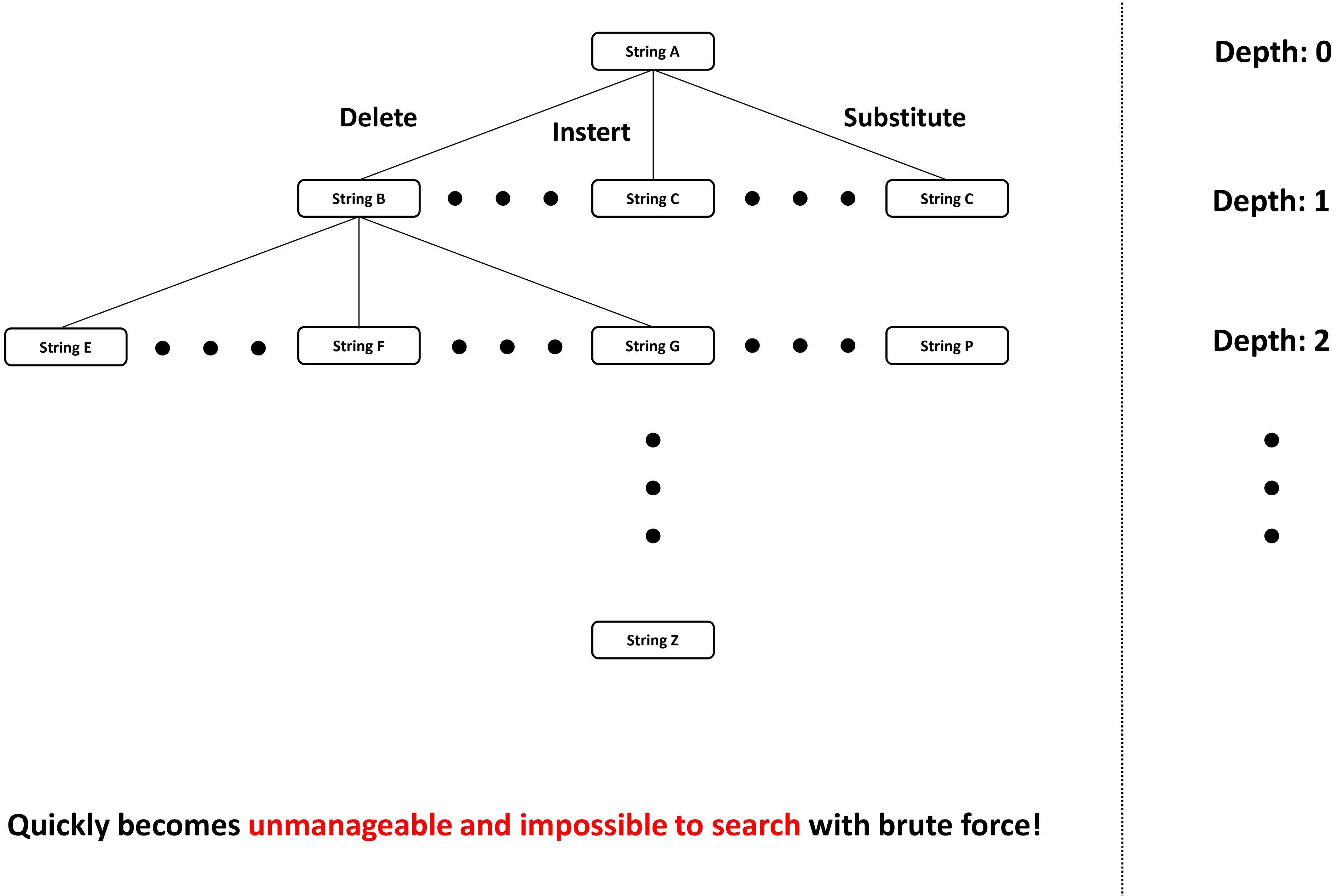
- **Initial state (root): the word we're transforming**
- **Operators / actions: insert, delete, substitute**
- **Goal state: the word we're trying to get to**
- **Path cost: what we want to minimize - the number of edits**

Edit Path

One of the edit paths (we want minimum # of edits):

```
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
```

Finding Minimum Edit Path /w Search



Quickly becomes unmanageable and impossible to search with brute force!

Minimum Edit Distance: Definition

- For two strings:
  - X of length n
  - Y of length m
- We define D(i, j)
  - the edit distance between X[1..i] and Y[1..j]
  - i.e., the first i characters of X and the first j characters of Y
  - The edit distance between X and Y is thus D(n, m)

MED: Dynamic Programming

- **Dynamic programming**: A tabular computation of D(n, m)
  - Solving problems by combining solutions to subproblems.
- Bottom-up approach
  - we compute D(i ,j) for small i, j
  - and then compute larger D(i, j) based on previously computed smaller values
    - i.e., compute D(i, j) for all i ($0 < i < n$) and j ($0 < j < m$)

Minimum Edit Distance: Pseudocode

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

  n ← LENGTH(source)
  m ← LENGTH(target)
  Create a distance matrix D[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]
```

Minimum Edit Distance: Pseudocode

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

  n ← LENGTH(source)
  m ← LENGTH(target)
  Create a distance matrix D[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]
```

Distance Matrix (m+1 x n+1): Setup

source string (m characters)	m	???	???	???	???	???	???	???	???	???
	m-1	???	???	???	???	???	???	???	???	???
	m-2	???	???	???	???	???	???	???	???	???
	m-3	???	???	???	???	???	???	???	???	???
	…	???	???	???	???	???	???	???	???	???
	…	???	???	???	???	???	???	???	???	???
	3	???	???	???	???	???	???	???	???	???
	2	???	???	???	???	???	???	???	???	???
	1	???	???	???	???	???	???	???	???	???
#	0	1	2	3	…	…	n-3	n-2	n-1	n
	#	target string (n characters)								

 # - empty string

Distance Matrix: Levenshtein Distance

source string (m characters)	m	???	???	???	???	???	???	???	???	???	???
	m-1	???	???	???	???	???	???	???	???	???	???
	m-2	???	???	???	???	???	???	???	???	???	???
	m-3	???	???	???	???	???	???	???	???	???	???
	...	???	???	???	???	???	???	???	???	???	???
	...	???	???	???	???	???	???	???	???	???	???
	3	???	???	???	???	???	???	???	???	???	???
	2	???	???	???	???	???	???	???	???	???	???
	1	???	???	???	???	???	???	???	???	???	???
#	0	1	2	3	...	...	n-3	n-2	n-1	n	
	#	target string (n characters)									

$$distance[i,j] = min \begin{cases} distance[i-1,j] + insertionCost(target_{i-1}) \\ distance[i-1,j-1] + substitutionCost(source_{j-1}, target_{i-1}) \\ distance[i,j-1] + deletionCost(source_{j-1}) \end{cases}$$

Distance Matrix: Levenshtein Distance

source string (m characters)	m	???	???	???	???	???	???	???	???	???
	m-1	???	???	???	???	???	???	???	???	???
	m-2	???	???	???	???	???	???	???	???	???
	m-3	???	???	???	???	???	???	???	???	???
	...	???	???	???	???	???	???	???	???	???
	...	???	???	???	???	???	???	???	???	???
	3	???	???	???	???	???	???	???	???	???
	2	???	???	???	???	???	???	???	???	???
	1	???	???	???	???	???	???	???	???	???
#	0	1	2	3	...	...	n-3	n-2	n-1	n
	#	target string (n characters)								

$$distance[col,row] = min\begin{cases} distance[col-1,row] + insertionCost(target_{col-1}) \\ distance[col-1,row-1] + substitutionCost(source_{row-1}, target_{col-1}) \\ distance[col,row-1] + deletionCost(source_{row-1}) \end{cases}$$

Distance Matrix: Levenshtein Distance

source string (m characters)	m	???	???	???	???	???	???	???	???	???
	m-1	???	???	???	???	???	???	???	???	???
	m-2	???	???	???	???	???	???	???	???	???
	m-3	???	???	???	???	???	???	???	???	???
	...	???	???	???	???	???	???	???	???	???
	...	???	???	???	???	???	???	???	???	???
	3	???	???	???	???	???	???	???	???	???
	2	???	???	???	???	???	???	???	???	???
	1	???	???	???	???	???	???	???	???	???
#	0	1	2	3	...	...	n-3	n-2	n-1	n
	#	target string (n characters)								

$$distance[i,j] = min\begin{cases} distance[i-1,j]+1 \\ distance[i-1,j-1]+2 \\ distance[i,j-1]+1 \end{cases}$$

2 if different characters
0 if same characters

Edit Distance Matrix: Calculations

source string (m characters)	m	???	???	???	???	???	???	???	???	???
	m-1	???	???	???	???	???	???	???	???	???
	m-2	???	???	???	???	???	???	???	???	???
	m-3	???	???	???	???	???	???	???	???	???
	…	???	???	???	???	???	???	???	???	???
	…	???	???	???	???	???	???	???	???	???
	3	???	???	???	???	???	???	???	???	???
	2	???	???	???	???	???	???	???	???	???
	1	???	???	???	???	???	???	???	???	???
#	0	1	2	3	…	…	n-3	n-2	n-1	n
	#	target string (n characters)								

□ ↑ □ - **insertion**

□ ↗ □ - **substitution**

□ ↑ □ - **deletion**

Minimum Edit Distance: Pseudocode

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

$n \leftarrow$ LENGTH(*source*)
$m \leftarrow$ LENGTH(*target*)
Create a distance matrix $D[n+1,m+1]$

\# *Initialization: the zeroth row and column is the distance from the empty string*
$\boxed{D[0,0] = 0}$
for each row i **from** 1 **to** n **do**
    $D[i,0] \leftarrow D[i\text{-}1,0] + \textit{del-cost}(source[i])$
for each column j **from** 1 **to** m **do**
    $D[0,j] \leftarrow D[0,j\text{-}1] + \textit{ins-cost}(target[j])$

\# *Recurrence relation:*
for each row i **from** 1 **to** n **do**
    for each column j **from** 1 **to** m **do**
        $D[i,j] \leftarrow \text{MIN}(\,D[i-1,j] + \textit{del-cost}(source[i]),$
            $D[i-1,j-1] + \textit{sub-cost}(source[i], target[j]),$
            $D[i,j-1] + \textit{ins-cost}(target[j]))$

\# *Termination*
return D[n,m]

Edit Distance Matrix: Initialization 1

n	???	???	???	???	???	???	???	???	???	???
o	???	???	???	???	???	???	???	???	???	???
i	???	???	???	???	???	???	???	???	???	???
t	???	???	???	???	???	???	???	???	???	???
n	???	???	???	???	???	???	???	???	???	???
e	???	???	???	???	???	???	???	???	???	???
t	???	???	???	???	???	???	???	???	???	???
n	???	???	???	???	???	???	???	???	???	???
i	???	???	???	???	???	???	???	???	???	???
#	$\boxed{0}$	???	???	???	???	???	???	???	???	???
	#	**e**	**x**	**e**	**c**	**u**	**t**	**i**	**o**	**n**

Minimum Edit Distance: Pseudocode

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

$n \leftarrow$ LENGTH(*source*)
$m \leftarrow$ LENGTH(*target*)
Create a distance matrix $D[n+1, m+1]$

\# *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**
    $D[i,0] \leftarrow D[i-1,0]$ + *del-cost*(*source*[i])
for each column j **from** 1 **to** m **do**
    $D[0,j] \leftarrow D[0,j-1]$ + *ins-cost*(*target*[j])

\# *Recurrence relation:*
for each row i **from** 1 **to** n **do**
    for each column j **from** 1 **to** m **do**
        $D[i,j] \leftarrow$ 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]

Edit Distance Matrix: Initialization 2

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

Minimum Edit Distance: Pseudocode

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

  n ← LENGTH(source)
  m ← LENGTH(target)
  Create a distance matrix D[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]
```

Edit Distance Matrix: Initialization 3

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**

Minimum Edit Distance: Pseudocode

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

  n ← LENGTH(source)
  m ← LENGTH(target)
  Create a distance matrix D[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]
```

Edit Distance Matrix: Populate

n	9	???	???	???	???	???	???	???	???	???
o	8	???	???	???	???	???	???	???	???	???
i	7	???	???	???	???	???	???	???	???	???
t	6	???	???	???	???	???	???	???	???	???
n	5	???	???	???	???	???	???	???	???	???
e	4	???	???	???	???	???	???	???	???	???
t	3	???	???	???	???	???	???	???	???	???
n	2	???	???	???	???	???	???	???	???	???
i	1	**0 + 2 = 2**	???	???	???	???	???	???	???	???
#	0	1	2	3	4	5	6	7	8	9
	#	**e**	**x**	**e**	**c**	**u**	**t**	**i**	**o**	**n**

$$distance[i,j] = min\begin{cases} distance[i-1,j] + insertionCost(target_{i-1}) \\ distance[i-1,j-1] + substitutionCost(source_{j-1}, target_{i-1}) \\ distance[i,j-1] + deletionCost(source_{j-1}) \end{cases}$$

Edit Distance Matrix: Populate

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**

$$distance[i,j] = min \begin{cases} distance[i-1,j] + insertionCost(target_{i-1}) \\ distance[i-1,j-1] + substitutionCost(source_{j-1}, target_{i-1}) \\ distance[i,j-1] + deletionCost(source_{j-1}) \end{cases}$$

Minimum Edit Path with Backtrace

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**

Idea: while populating, add "pointers" (↓←↙) to indicate which cell did we come from. Use pointers to "backtrace" by following the minimum edit path.

Minimum Edit Path with Backtrace

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**

↓←↙ - which cell did we come from?

red - minimum edit cost

Minimum Edit Path with Backtrace

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**

↓←↙ - which cell did we come from?

red - minimum edit cost

Minimum Edit Path with Backtrace

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**

↓←↙ - which cell did we come from?

red - minimum edit cost

Minimum Edit Path with Backtrace

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**

↓←↙ - which cell did we come from?

red - minimum edit cost

Minimum Edit Path with Backtrace

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**

↓←↙ - which cell did we come from?

red - minimum edit cost

Minimum Edit Path with Backtrace

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**

↓←↙ - which cell did we come from?

red - minimum edit cost

Minimum Edit Path with Backtrace

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**

↓←↙ - which cell did we come from?

red - minimum edit cost

Minimum Edit Path with Backtrace

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**

↓←↙ - which cell did we come from?

red - minimum edit cost

Minimum Edit Path with Backtrace

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**

↓←↙ - which cell did we come from?

red - minimum edit cost

Minimum Edit Path with Backtrace

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**

↓←↙ - which cell did we come from?

red - minimum edit cost

Minimum Edit Path with Backtrace

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**

↓←↙ - which cell did we come from?

red - minimum edit cost

Minimum Edit Path with Backtrace

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**

Final minimum edit path.

Time and Space Complexity

- **Time:**

$$O(n * m)$$

- **Space:**

$$O(n * m)$$

- **Backtrace time complexity:**

$$O(n + m)$$

Weighted Edit Distance

- **Why would we add weights to the computation?**
- **Spell Correction:**
  - **some letters are more likely to be mistyped than others**
- **Biology:**
  - **certain kinds of deletions or insertions are more likely than others**

Weighted Edit Distance

Confusion matrix for spelling errors:

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

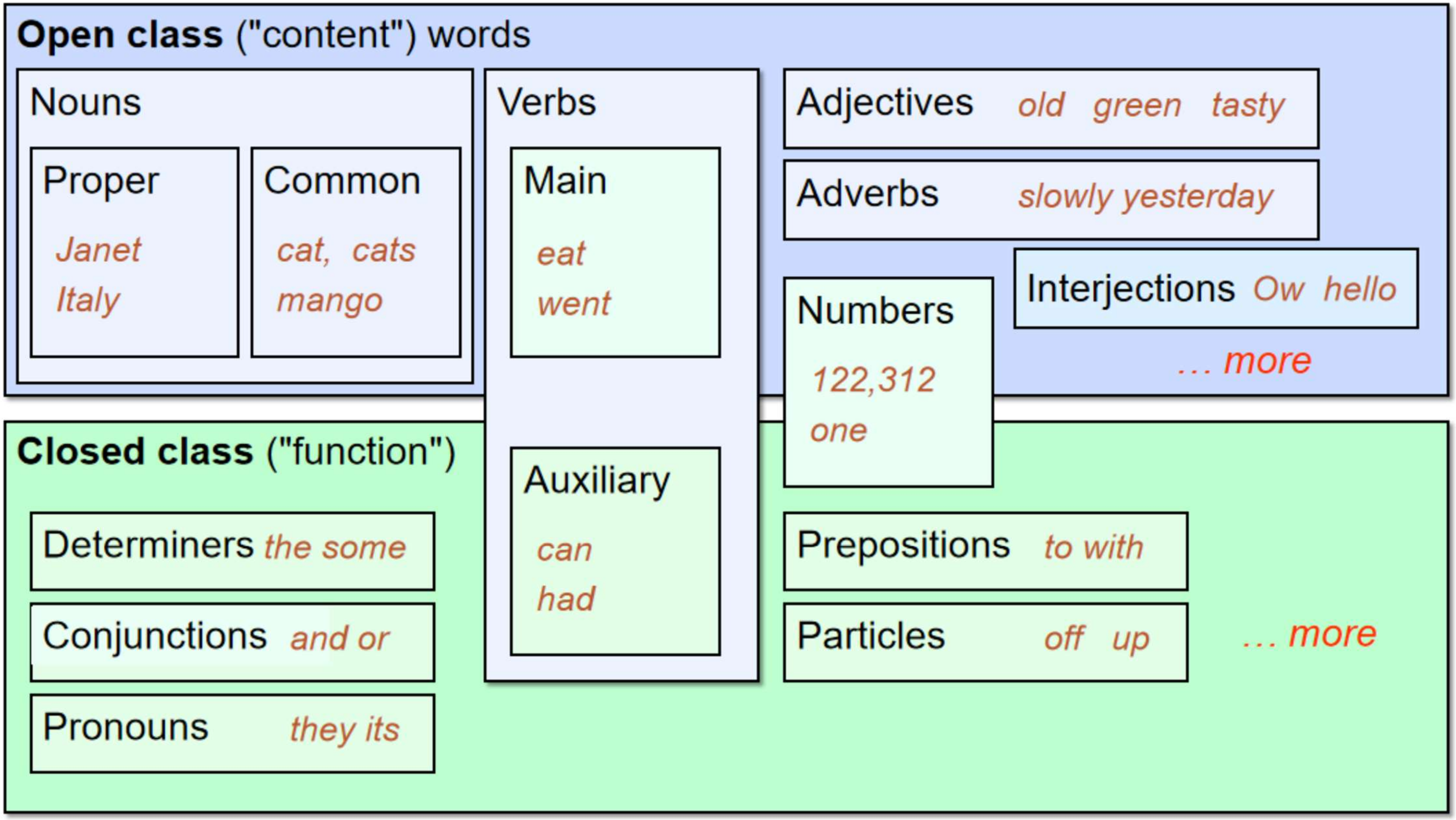
Parts of Speech

- **Idea:**
  - **classify words according to their grammatical categories**
- **Categories = part of speech, word classes, POS, POS tags**
- **Basic categories / tags:**
  - **noun, verb, pronoun, preposition, adverb, conjunction, participle, article**

Parts of Speech: Closed vs. Open

- **Closed class:**
  - **relatively fixed set - new members rarely added**
  - **usually function words: short, frequent words with grammatical function:**
    - **determiners:** *a*, *an*, *the*
    - **pronouns:** *she*, *he*, *I*
    - **prepositions:** *on*, *under*, *over*, *near*, ***by***
- **Open class:**
  - **word sets where new members are constantly created**
  - **usually content words: nouns, verbs, adjectives, adverbs**
  - **new words | examples: nouns (*iPhone*), verbs (*to google*)**

Parts of Speech: Closed vs. Open



Parts of Speech Tagging

- **Assigning a part-of-speech (POS) to each word in a text.**
- **Words often have more than one POS.**
  - **example:** *book*
    - **VERB:** ***Book*** *that flight*
    - **NOUN:** *Hand me that* ***book***

Sample Tagged Sentence

There/PRO were/VERB 70/NUM children/NOUN there/ADV ./PUNC

Preliminary/ADJ findings/NOUN were/AUX reported/VERB in/ADP today/NOUN ’s/PART New/PROPN England/PROPN Journal/PROPN of/ADP Medicine/PROPN

Parts of Speech: Tagset Example

Parts of Speech in the Universal Dependencies tagset

	Tag	Description	Example
Open Class	**ADJ**	Adjective: noun modifiers describing properties	*red, young, awesome*
	ADV	Adverb: verb modifiers of time, place, manner	*very, slowly, home, yesterday*
	NOUN	words for persons, places, things, etc.	*algorithm, cat, mango, beauty*
	VERB	words for actions and processes	*draw, provide, go*
	PROPN	Proper noun: name of a person, organization, place, etc..	*Regina, IBM, Colorado*
	INTJ	Interjection: exclamation, greeting, yes/no response, etc.	*oh, um, yes, hello*
Closed Class Words	**ADP**	Adposition (Preposition/Postposition): marks a noun's spacial, temporal, or other relation	*in, on, by, under*
	AUX	Auxiliary: helping verb marking tense, aspect, mood, etc.,	*can, may, should, are*
	CCONJ	Coordinating Conjunction: joins two phrases/clauses	*and, or, but*
	DET	Determiner: marks noun phrase properties	*a, an, the, this*
	NUM	Numeral	*one, two, first, second*
	PART	Particle: a preposition-like form used together with a verb	*up, down, on, off, in, out, at, by*
	PRON	Pronoun: a shorthand for referring to an entity or event	*she, who, I, others*
	SCONJ	Subordinating Conjunction: joins a main clause with a subordinate clause such as a sentential complement	*that, which*
Other	**PUNCT**	Punctuation	; , ()
	SYM	Symbols like $ or emoji	$, %
	X	Other	asdf, qwfg

Parts of Speech: Tagset Example

Penn Treebank Parts-of-speech tags:

Tag	Description	Example	Tag	Description	Example	Tag	Description	Example
CC	coord. conj.	*and, but, or*	NNP	proper noun, sing.	*IBM*	TO	"to"	*to*
CD	cardinal number	*one, two*	NNPS	proper noun, plu.	*Carolinas*	UH	interjection	*ah, oops*
DT	determiner	*a, the*	NNS	noun, plural	*llamas*	VB	verb base	*eat*
EX	existential 'there'	*there*	PDT	predeterminer	*all, both*	VBD	verb past tense	*ate*
FW	foreign word	*mea culpa*	POS	possessive ending	*'s*	VBG	verb gerund	*eating*
IN	preposition/ subordin-conj	*of, in, by*	PRP	personal pronoun	*I, you, he*	VBN	verb past partici-ple	*eaten*
JJ	adjective	*yellow*	PRP$	possess. pronoun	*your, one's*	VBP	verb non-3sg-pr	*eat*
JJR	comparative adj	*bigger*	RB	adverb	*quickly*	VBZ	verb 3sg pres	*eats*
JJS	superlative adj	*wildest*	RBR	comparative adv	*faster*	WDT	wh-determ.	*which, that*
LS	list item marker	*1, 2, One*	RBS	superlatv. adv	*fastest*	WP	wh-pronoun	*what, who*
MD	modal	*can, should*	RP	particle	*up, off*	WP$	wh-possess.	*whose*
NN	sing or mass noun	*llama*	SYM	symbol	*+,%, &*	WRB	wh-adverb	*how, where*

Parts of Speech Tagging: Motivation

- Can be useful for other NLP tasks
  - Parsing: POS tagging can improve syntactic parsing
  - MT: reordering of adjectives and nouns (say from Spanish to English)
  - Sentiment or affective tasks: may want to distinguish adjectives or other POS
  - Text-to-speech (how do we pronounce "*lead*" or "***object***"?)
- Or linguistic or language-analytic computational tasks
  - Need to control for POS when studying linguistic change like creation of new words, or meaning shift
  - Or control for POS in measuring meaning similarity or difference