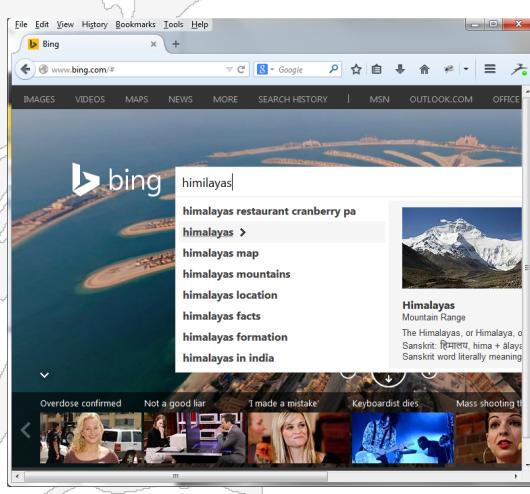
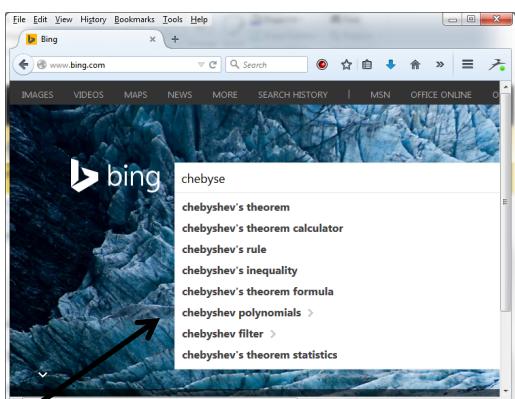
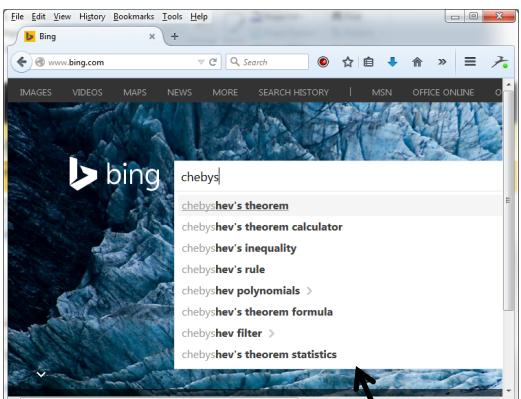
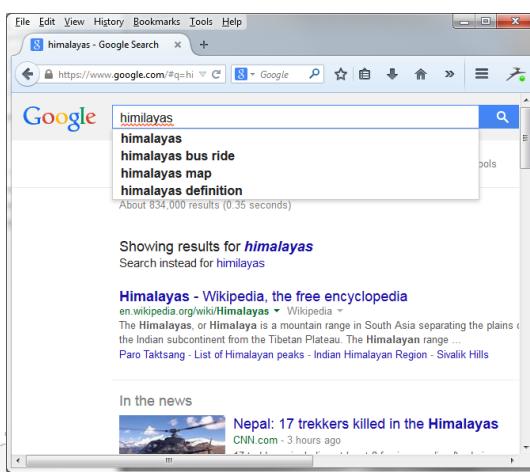
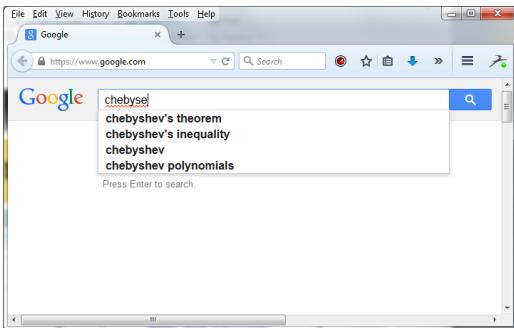
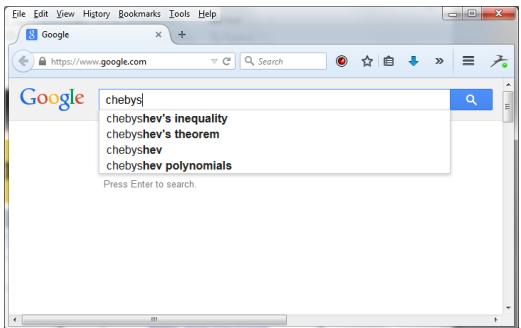


Spell Checking and Correction

Some Google/Bing Examples

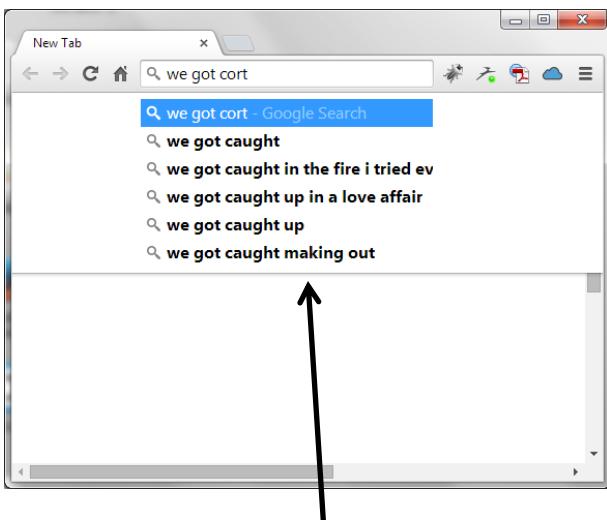


Bing uses the autocomplete options for spelling correction
 but there is no red underline

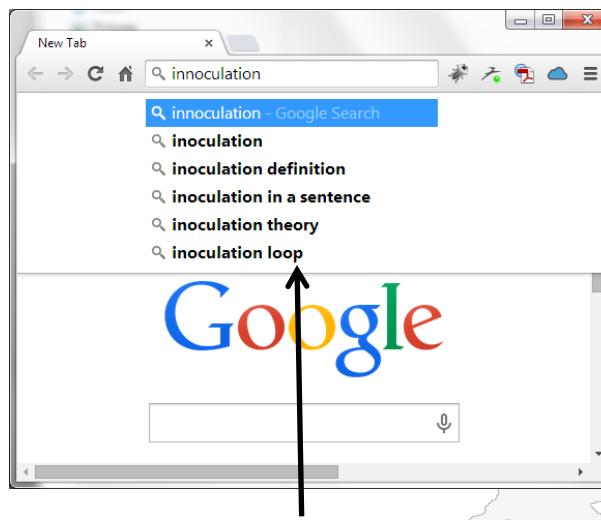
Copyright Ellis Horowitz 2011-2018

Himilayas

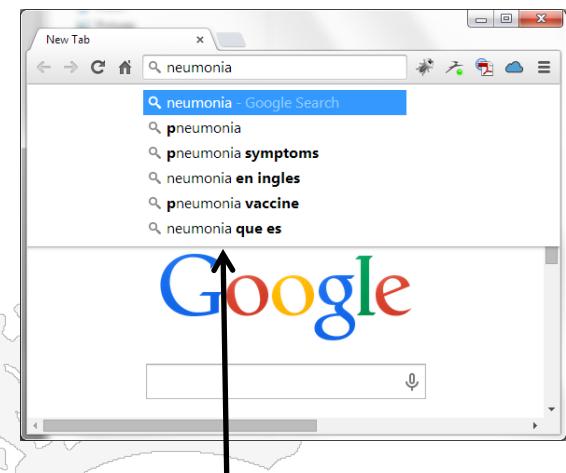
More Examples



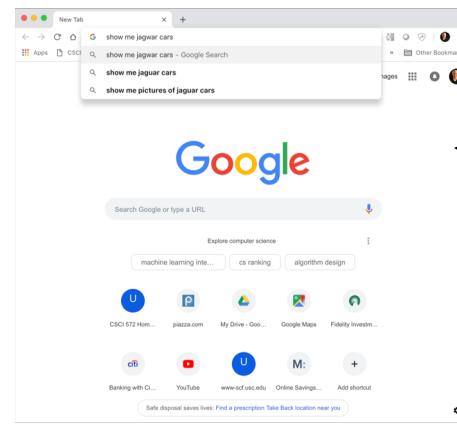
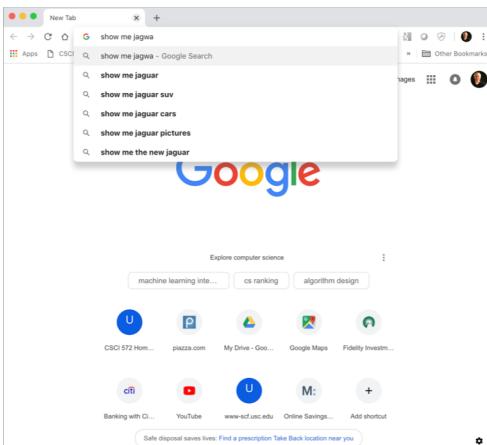
easy for people, but harder for computers
to correct, use n-grams



easy for a computer to correct as the
word is in the database, hard for a person



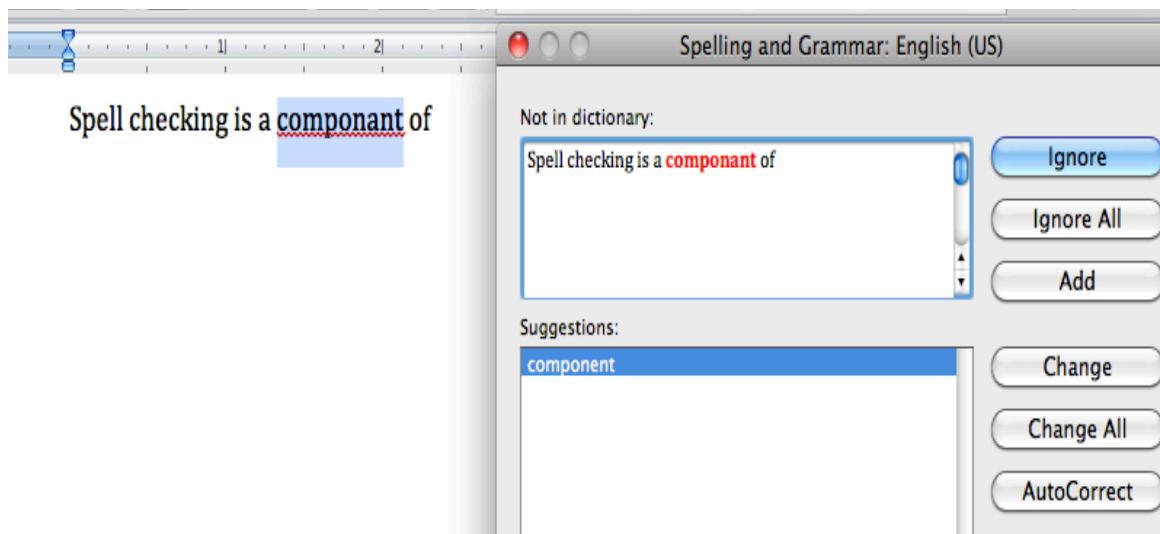
computer needs to both
identify the error and correct
the misspelling



Google uses most likely terms
for autocomplete and leaves the
user's misspelling

Other Places where Spelling Correction is Critical

1. Word processing



2. Smartphone input



Rates of Spelling Errors

Error rates vary depending upon the application

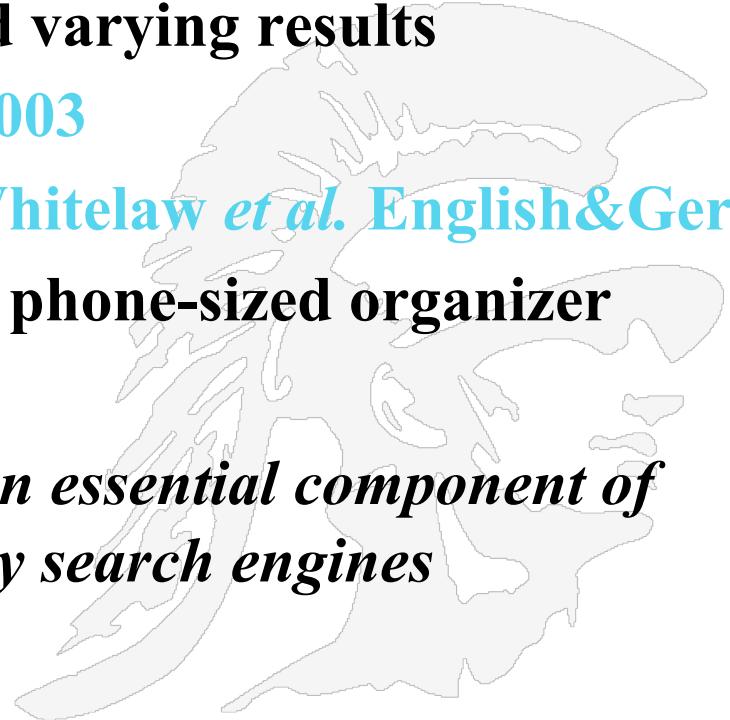
- Typing is very error prone, and especially difficult on smartphones
- Different studies have produced varying results

26%: Web queries [Wang et al. 2003](#)

13%: Retyping, no backspace: [Whitelaw et al. English&German](#)

7%: Words corrected retyping on phone-sized organizer

So seamless spelling correction is an essential component of information retrieval and, especially search engines



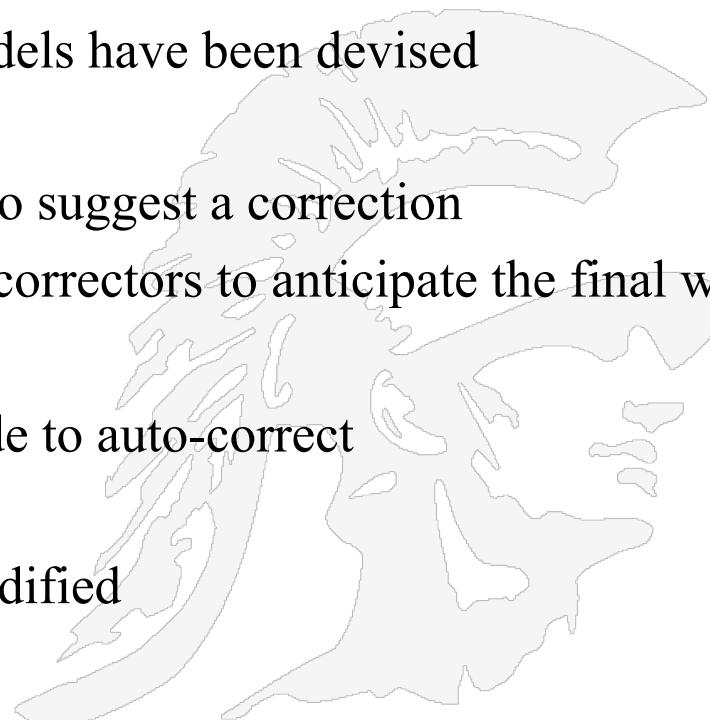
The Two Main Spelling Tasks

1. Spelling Error *Detection*

- Obviously we need a big dictionary
- Using context may be necessary
 - To do this spelling error models have been devised

2. Spelling Error *Correction*

- Web search engines always try to suggest a correction
- Autocomplete requires spelling correctors to anticipate the final word
 - Fast response time is required
- Word and PowerPoint have mode to auto-correct
 - set as the default
 - the spell dictionary can be modified



Three Types of Spelling Errors

1. Non-word errors

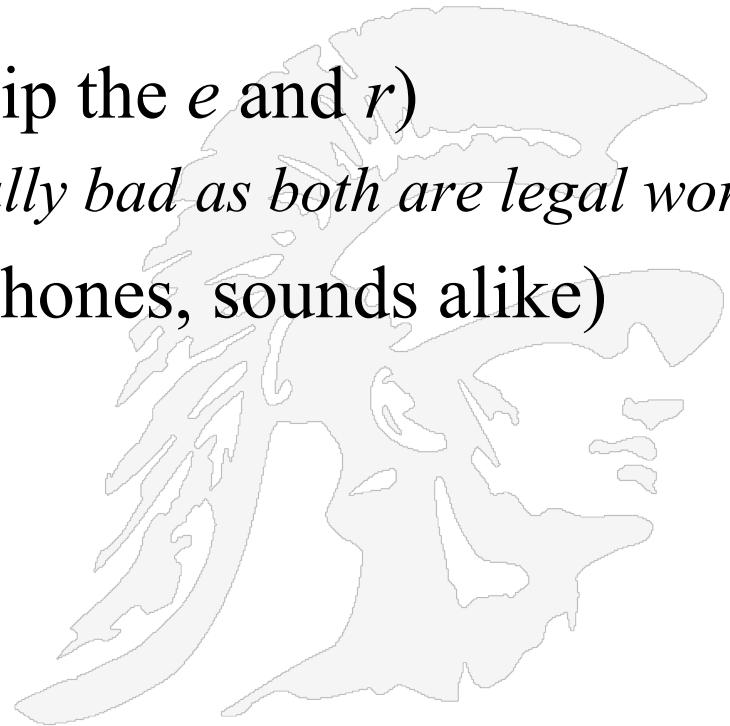
- *graffe* → *giraffe*

2. Typographical errors (flip the *e* and *r*)

- *three* → *there* (*especially bad as both are legal words*)

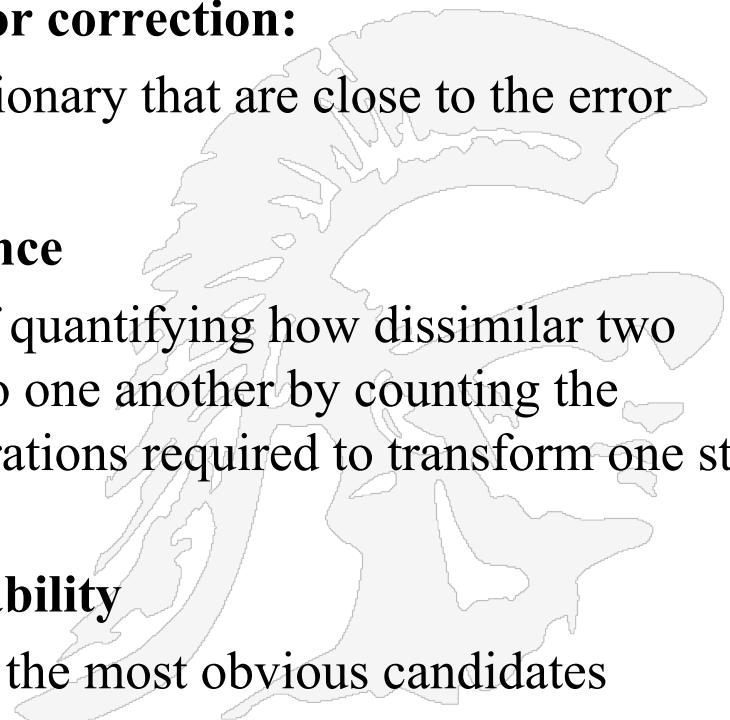
3. Cognitive errors (homophones, sounds alike)

- *piece* → *peace*,
- *too* → *two*
- *your* → *you're*



Non-Word Spelling Errors

- **Non-word spelling error detection:**
 - Any word not in a *dictionary* is presumed to be an error
 - The larger the dictionary the better
- **Approach to non-word spelling error correction:**
 - Generate candidates from the dictionary that are close to the error
 - **How do we do this?**
 - **Shortest weighted edit distance**
 - **Edit distance** is a way of quantifying how dissimilar two strings (e.g., words) are to one another by counting the minimum number of operations required to transform one string into the other
 - **Highest noisy channel probability**
 - use probabilities to select the most obvious candidates



Causes of Misspellings

Cause	Misspelling	Correction
typing quickly	exxit misspell	exit misspell
keyboard adjacency	importamt	important
inconsistent rules	conceive concierge	conceive concierge
ambiguous word breaking	silver light	silverlight
new words	kinnect	kinect

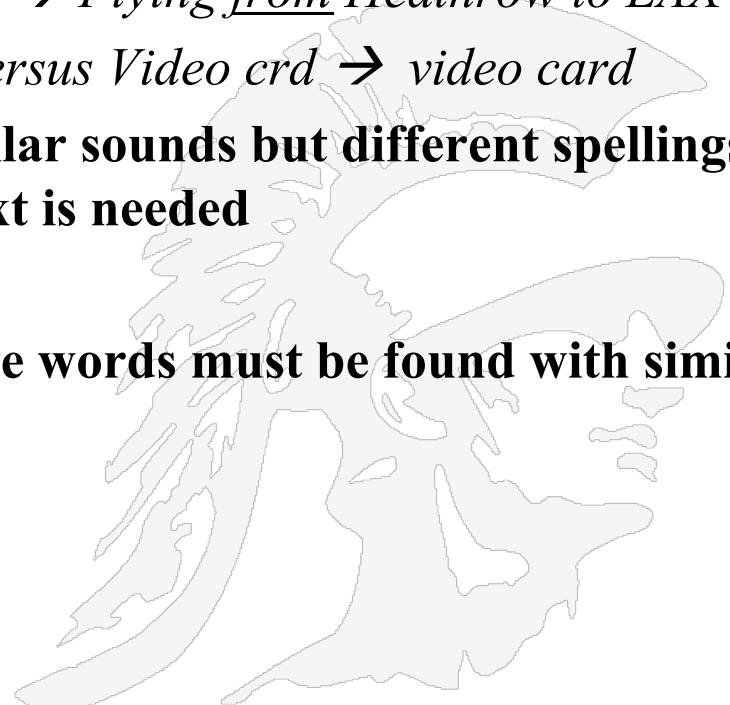


e before i
i before e


According to Cucerzan and Brill, **more than 10% of search engine queries are misspelled**
"Spelling Correction as an iterative process that exploits the collective knowledge of web users"
<http://www-scf.usc.edu/~csci572/papers/Cucerzan.pdf> (advocates using query logs to guess the correct spelling)

Spelling Errors Needing Context

- Three types of misspellings
 1. consider whether the surrounding words “make sense” for your candidate set, e.g.
 - *Flying form Heathrow to LAX* → *Flying from Heathrow to LAX*
 - *Power crd* → *power cord* versus *Video crd* → *video card*
 2. For candidate words with similar sounds but different spellings and different meanings, context is needed
 - e.g. *there, their*
 3. To resolve the above, candidate words must be found with similar *pronunciations*
 - *use the Soundex algorithm*



More Challenges for Identifying Spelling Errors

- Some additional challenges

- 4. Allow for insertion of a space or hyphen

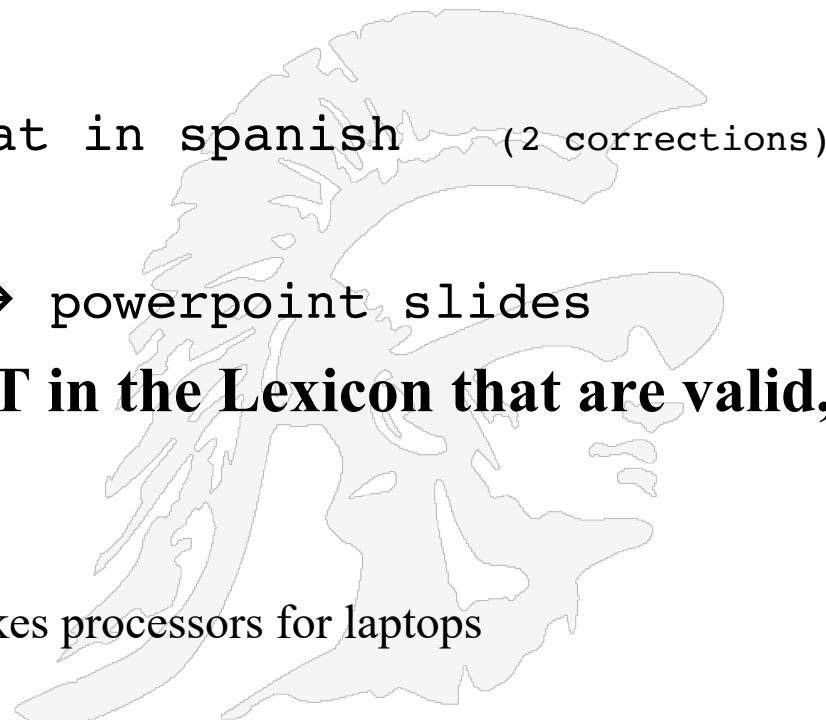
- thisidea → this idea
 - inlaw → in-law
 - chat inspanich → chat in spanish (2 corrections)

- 5. Allow deletion of a space

- power point slides → powerpoint slides

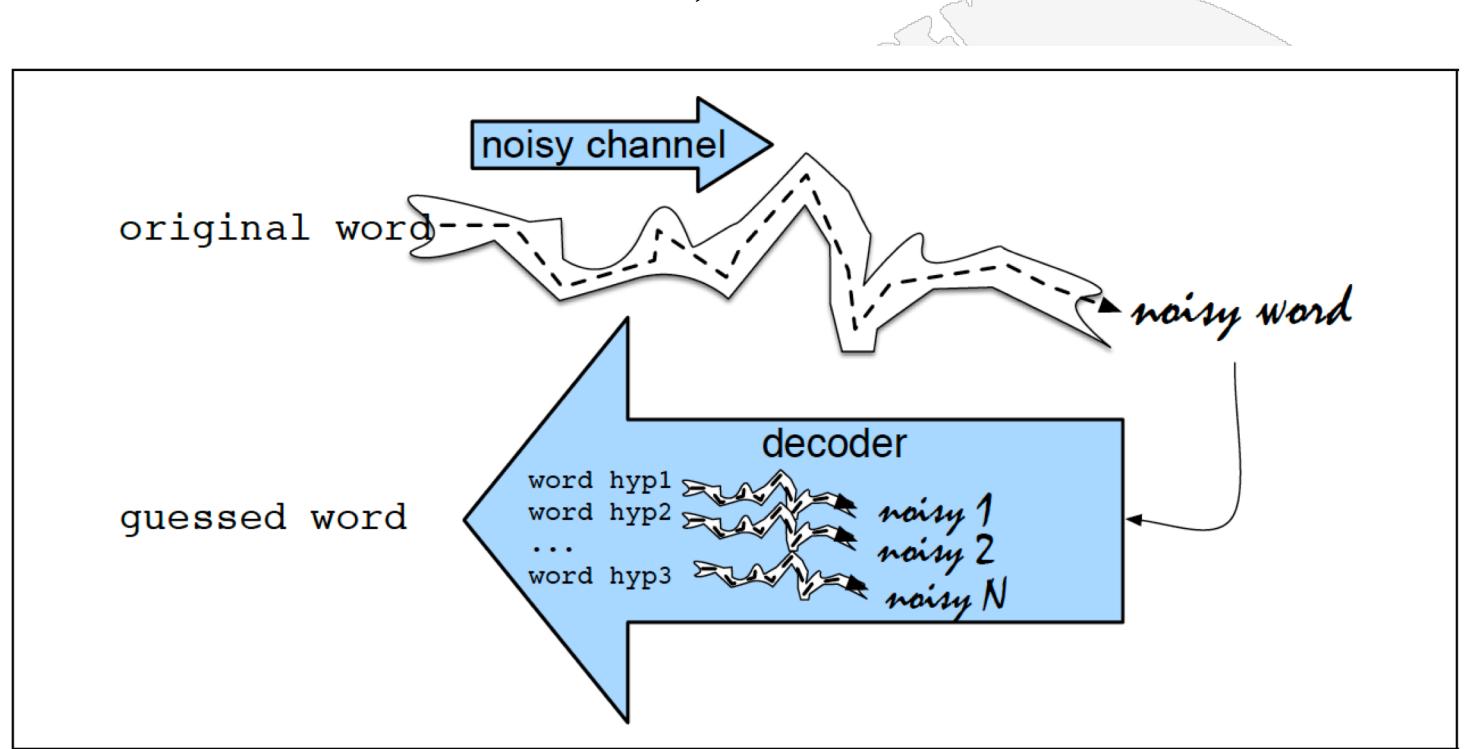
- 6. Watch out for words NOT in the Lexicon that are valid, e.g.

- amd processors
 - AMD is a company that makes processors for laptops
 - Context is needed



The Noisy Channel Model

- This model suggests treating the misspelled word as if a correctly spelled word has been distorted by being passed through a noisy communication channel
- Noise in this case refers to substitutions, insertions or deletions of letters



Bayesian Inference Implies We Can Use Previous Combinations to Predict the Correct Word

- We see an observation x (a misspelled word) and our job is to find the correct word w that generated this misspelled word
- Out of all possible words in the vocabulary V we want to find the word w such that $P(w|x)$ is highest. We use the hat notation \hat{w} to mean “our estimate of the correct word”

$$\hat{w} = \operatorname{argmax}_{w \in V} P(w|x)$$

- Out of all words in the vocabulary, we want the particular word that maximizes the right-hand side $P(w|x)$
- *Bayes Rule:*

$$P(a|b) = \frac{P(b|a)P(a)}{P(b)}$$

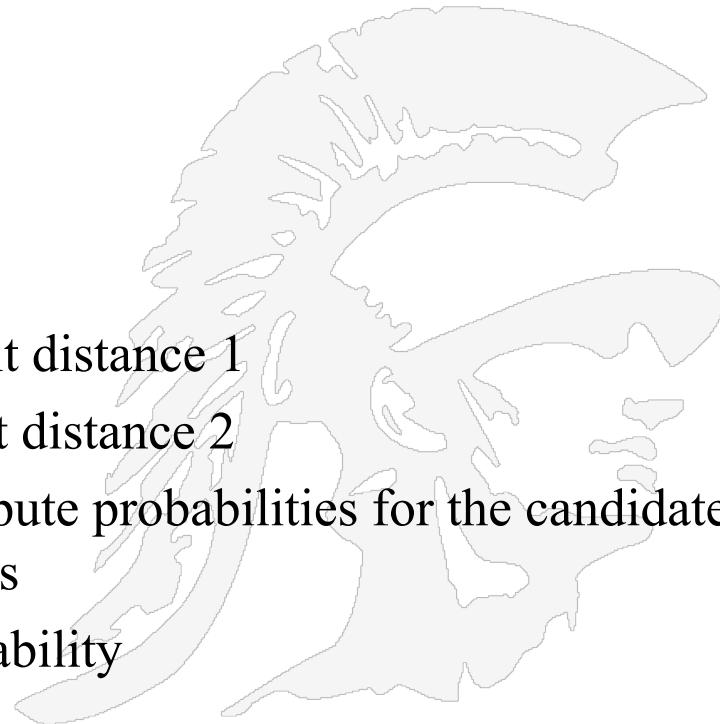
implies $\hat{w} = \operatorname{argmax}_{w \in V} \frac{P(x|w)P(w)}{P(x)}$ and simplifying gives $\hat{w} = \operatorname{argmax}_{w \in V} P(x|w)P(w)$

The probability of a given b is equal to the probability of b given a , times the probability of a , divided by the probability of b

Note: $P(x)$ is a constant so we can ignore it

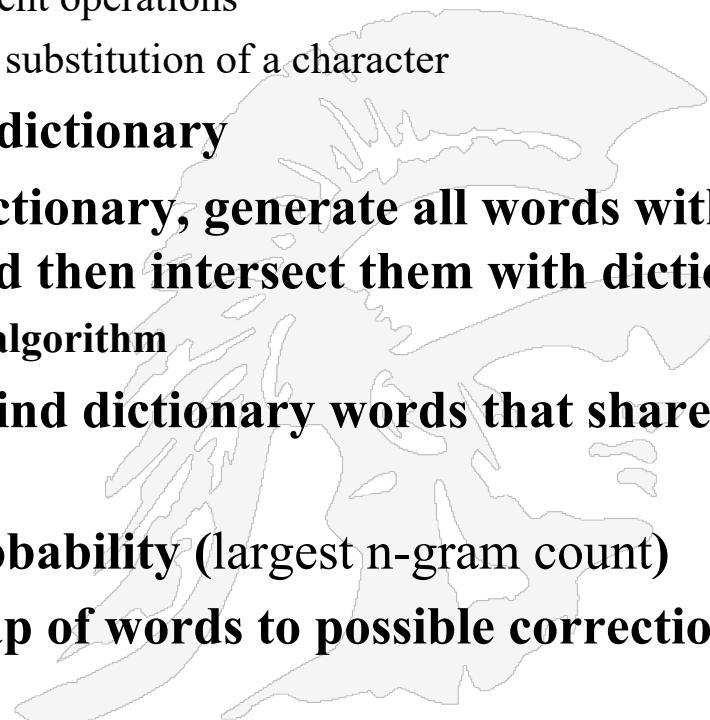
The Basic Spelling Correction Algorithm

1. **Initial step:** Create a dictionary and encode it for fast retrieval
2. When a query is submitted, the spell checker examines each word and looks for possible character edits, namely
 - insertions,
 - deletions,
 - substitutions, and occasionally
 - transpositions
 - **Observation:**
 - 80% of errors are within edit distance 1
 - Almost all errors within edit distance 2
3. Take the output of step 2 and compute probabilities for the candidates using previously identified n-grams
4. Select the result with highest probability



The Basic Spelling Correction Algorithm Refined

- **Edit distance** is a way of quantifying how dissimilar two strings (e.g. words) are to one another by counting the minimum number of operations required to transform one string into the other
 - different algorithms assume slightly different operations
 - e.g., Levenshtein uses: removal, insertion, substitution of a character
1. **Check each query term against the dictionary**
 2. **For each term NOT found in the dictionary, generate all words within edit distance $\leq k$ (e.g., $k = 1$ or 2) and then intersect them with dictionary**
 - Compute them fast with a Levenshtein algorithm
 3. **Use a character n -gram index and find dictionary words that share “most” n -grams with word**
 4. **Select the word with the highest probability (largest n-gram count)**
 5. **For speed, have a pre-computed map of words to possible corrections**



Use Edit Distance To Produce Candidate Corrections

Input	Candidate Correction	Correct Letter	Error Letter	correction Type
acress	actress	t	-	insertion
acress	cress	-	a	deletion
acress	caress	ca	ac	transposition
acress	access	c	r	substitution
acress	across	o	e	substitution
acress	acres	-	s	deletion

Six words within 1 of acress

Context check is necessary to choose the appropriate word

(try this yourself in Google/Bing)

For the word "acress" there are six dictionary words all within edit distance 1

Now Apply Probabilities

- We now need to compute the prior probability of each occurrence
- We can do this using unigrams, bigrams, trigrams, etc
- Using the Corpus of Contemporary English, 404,253,213 words we get the following
- *Across* is the most likely choice, followed by

word	Frequency of word	$P(w)$
actress	9,321	.0000230573
cress	220	.0000005442
caress	686	.0000016969
access	37,038	.0000916207
→ across	120,844	.0002989314
acres	12,874	.0000318463

For fun try:
I need a cress to . . .
I love the a cress . . .
a kiss and a a cress . . .

"across" is the
most likely
correction →

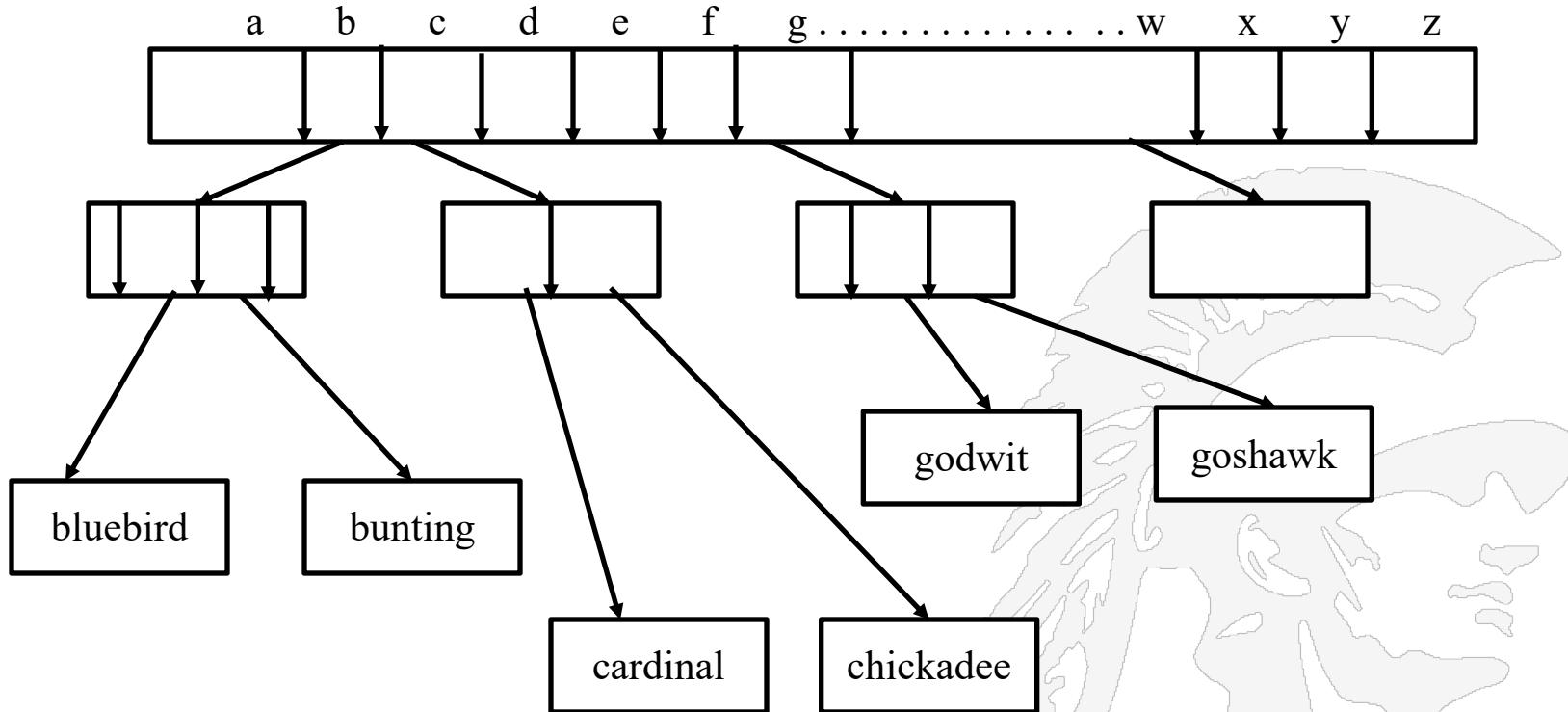
The Spelling Correction Dictionary and Autocomplete

- The search for corrections is carried out from left-to-right
 - At each point, a partial hypothesis is expanded with every character which could follow the partial hypothesis and lead to one of the known tokens (the user input is always allowed as an output hypothesis).
 - Thus the branching factor controls the amount of time required to search for spelling corrections.
- The terms of the lexicon must be stored in a data structure that affords efficient *prefix matching*
 - Often a trie data structure is used

The Spelling Correction Dictionary

Example of a Trie

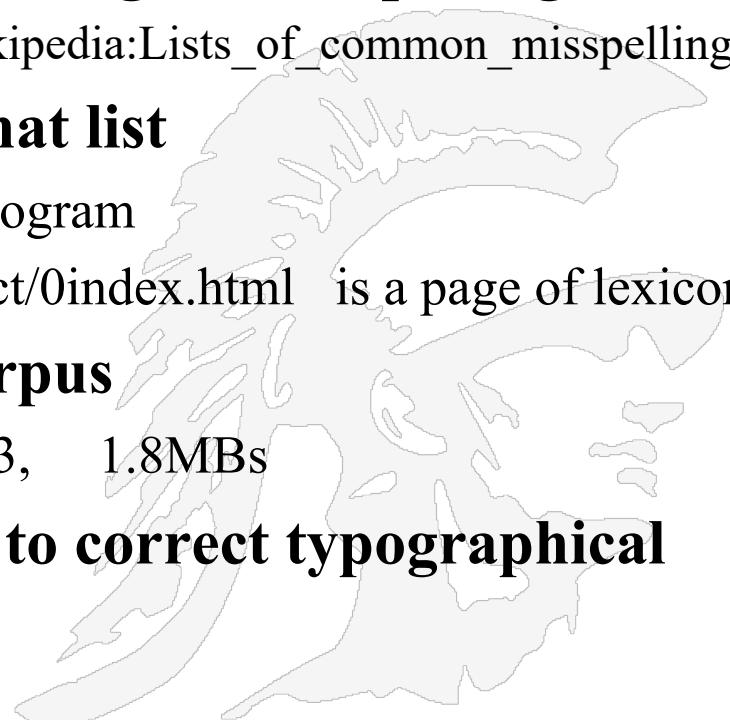
- a prefix tree (sometimes called a trie from the word retrieval) is a tree of degree ≥ 2 in which the branching at any level is determined by a portion of the key



branch nodes take you down the tree to element nodes; At any stage one is pointing at all keyword matches that contain the same prefix; Computing time for retrieval is $O(m)$ where m is the length of the string, at the expense of increased storage

The Spelling Correction Dictionary Error Test Sets

- To enhance a lexicon one can include a table of common misspellings
- there are many possible spelling error test sets, e.g.
 - Wikipedia's list of common English misspelling
 - https://en.wikipedia.org/wiki/Wikipedia:Lists_of_common_misspellings
 - Aspell filtered version of that list
 - <http://aspell.net/> is the spell program
 - <ftp://ftp.gnu.org/gnu/aspell/dict/0index.html> is a page of lexicons
 - Birkbeck spelling error corpus
 - <http://ota.ahds.ac.uk/desc/0643>, 1.8MBs
- These sets are primarily used to correct typographical errors



Using N-Grams For Spelling Correction

- An ***n*-gram model** is a type of probabilistic language model for predicting the next item in a sequence
- Two benefits of *n*-gram models (and algorithms that use them) are simplicity and scalability – with larger *n*, a model can store more context with a well-understood space–time tradeoff, enabling small experiments to scale up efficiently
- **Sample of 3-gram sequences**
- ceramics collectables fine (130)
- ceramics collected by (52)
- ceramics collectible pottery (50)
- ceramics collectibles cooking (45)

Sample of 4-gram sequences and # of times appeared

- serve as the incoming (92)
- serve as the incubator (99)
- serve as the independent (794) ←
- serve as the index (223)
- serve as the indication (72)
- serve as the indicator (120) ↑

a query such as "serve as the indapendant" would match the above

Google's N-Gram Data

- Google has collected and uses a great deal of N-gram data
- Google is using the Linguistics Data Consortium to distribute more than one trillion words they have extracted from public web pages
- Below is a statistical summary of the data they are distributing

File sizes: approx. 24 GB compressed (gzip'ed) text files

Number of tokens: 1,024,908,267,229

Number of sentences: 95,119,665,584

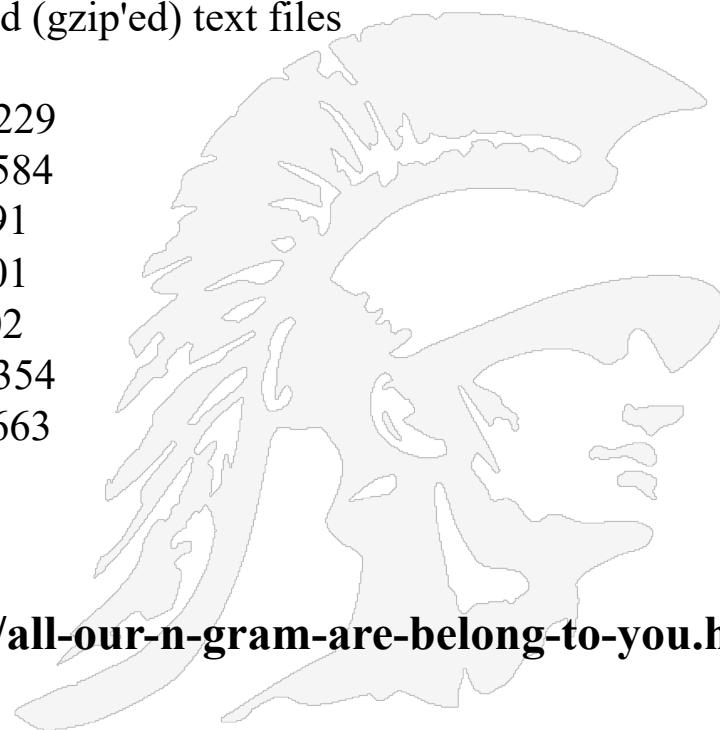
Number of unigrams: 13,588,391

Number of bigrams: 314,843,401

Number of trigrams: 977,069,902

Number of fourgrams: 1,313,818,354

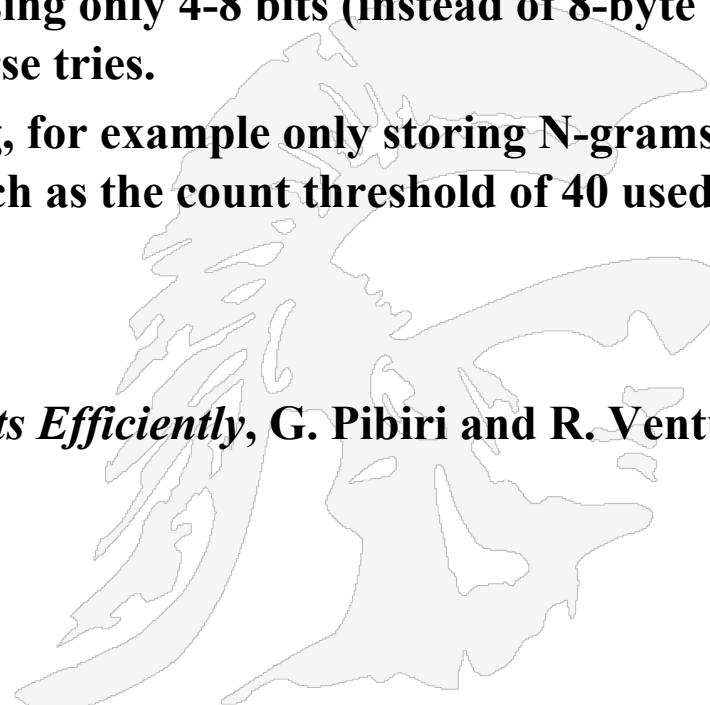
Number of fivegrams: 1,176,470,663



<http://googleresearch.blogspot.com/2006/08/all-our-n-gram-are-belong-to-you.html>

Handling N-Gram Data

- Efficiency considerations are important when building language models that use such large sets of N-grams.
- Rather than store each word as a string, it is generally represented in memory as a 64-bit hash number, with the words themselves stored on disk.
- Probabilities are generally quantized using only 4-8 bits (instead of 8-byte floats), and N-grams are stored in reverse tries.
- N-grams can also be shrunk by pruning, for example only storing N-grams with counts greater than some threshold (such as the count threshold of 40 used for the Google N-gram release)
- See *Handling Massive N-Gram Datasets Efficiently*, G. Pibiri and R. Venturini



Applying the N-Gram Model to Spelling Correction

- Let's say you are using 4-grams to calculate the probability of a next word in text.
 - You have "this is a very" followed by "sunny".
 - But assume "sunny" does not occur in the n-gram set that includes "this is a very"
 - Implication: so for the 4-gram model "sunny" has probability 0,
- Back-off means you go back to a $n-1$ -gram level to calculate the probabilities when you encounter a word with probability = 0.
 - So in the above you will use a 3-gram model to calculate the probability of "sunny" in the context "is a very".
- Whenever you go back one level you multiply the odds by 0.4. So if sunny exists in the 3-gram model the probability would be
$$0.4 * P("sunny" | "is a very")$$

The Stupid Back-off Algorithm

1. If a higher-order N-gram has a zero count, we simply backoff to a lower order N-gram, weighed by a fixed (context-independent) weight
 2. The backoff terminates in the unigram
- See Wikipedia's description of the Back-off Model

A Complete Spelling Correction Program



Peter Norvig's Spelling Corrector written in Python

```

import re, collections
def words(text): return re.findall('[a-z]+', text.lower())
def train(features):
    model = collections.defaultdict(lambda: 1)
    for f in features:
        model[f] += 1
    return model
NWORDS = train(words(file('big.txt').read()))
alphabet = 'abcdefghijklmnopqrstuvwxyz'
def edits1(word):
    splits      = [(word[:i], word[i:]) for i in range(len(word) + 1)]
    deletes     = [a + b[1:] for a, b in splits if b]
    transposes = [a + b[1] + b[0] + b[2:] for a, b in splits if len(b)>1]
    replaces   = [a + c + b[1:] for a, b in splits for c in alphabet if b]
    inserts    = [a + c + b      for a, b in splits for c in alphabet]
    return set(deletes + transposes + replaces + inserts)

def known_edits2(word):
    return set(e2 for e1 in edits1(word) for e2 in edits1(e1) if e2 in NWORDS)

def known(words): return set(w for w in words if w in NWORDS)

def correct(word):
    candidates = known([word]) or known(edits1(word)) or known_edits2(word) or [word]
    return max(candidates, key=NWORDS.get)

```

Correct takes a word
and returns a likely
correct form of the word:

```

correct('speling')
Spelling
correct('korrechter')
corrector

```

<http://norvig.com/spell-correct.html>

How the Program Works

- The algorithm and its description can be found at
- <http://norvig.com/spell-correct.html>
- The file `big.txt` contains a million words
 - It includes books from Project Gutenberg, Wiktionary, British National Corpus
- Extract the individual words (function `words` converts everything to lower case)
- *The* and *the* are the same, but “don’t” is seen as “don” and “t”
- The program counts how many times each word occurs using function `train`
- `NWORDS[w]` holds a count of how many times the word `w` has been seen
- For words that are not in our set we set their occurrence to a default, non-zero value of 1 using the Python hash table statement `collections.defaultdict`
- edit distance is the number of changes it would take to turn one word into another
- An edit can be one of: {deletion, transposition, alteration, insertion}
- The 7 lines that end in `return set(deletes + transposes + replaces + inserts)` is a function that returns a set of all words `c` that are one edit away from `w`
 - This can be a large set
- The literature on spelling correction claims that 80-95% of spelling errors are an edit distance of 1 from the target

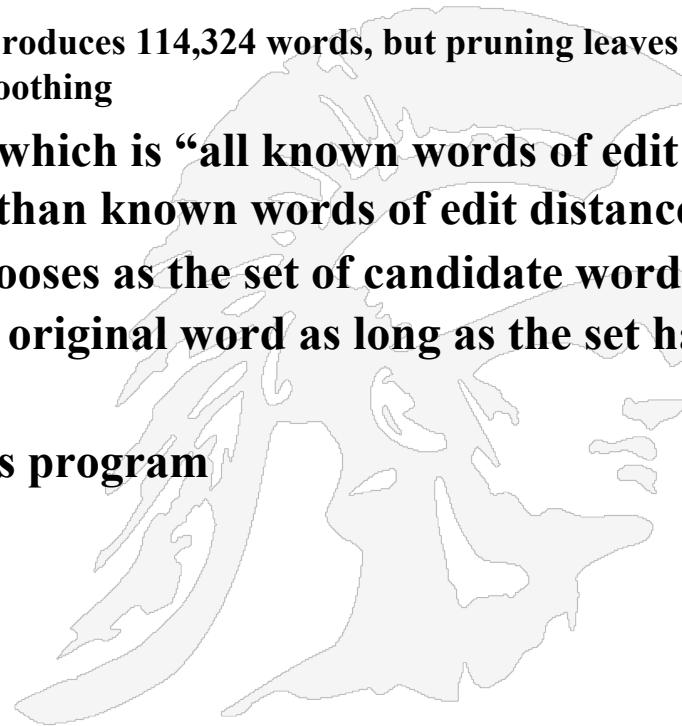
How the Program Works (cont'd)

- To compute edit distance 2, just apply edits1 to all the results of edits1

```
Def edit2(word) :
```

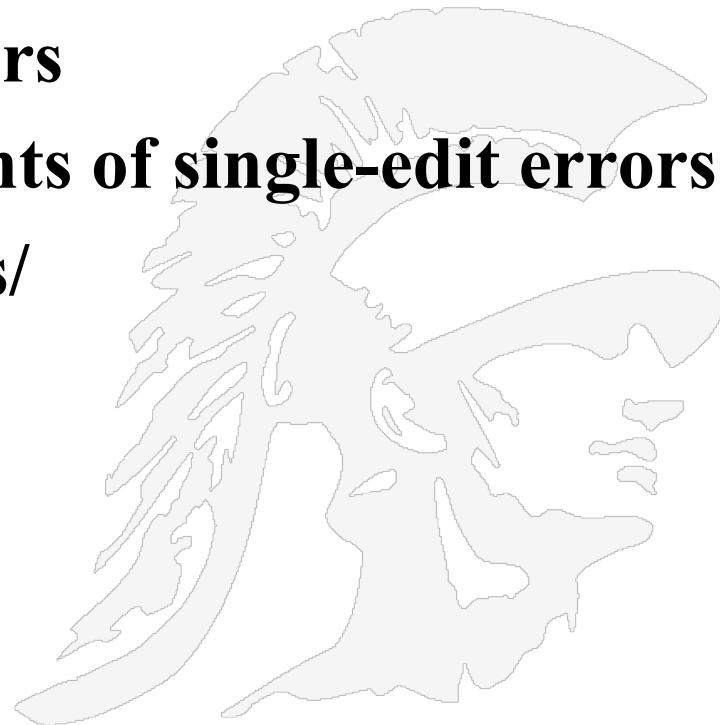
```
    return set(e2 for e1 in edits1(word) for e2 in edits1(e1))
```

- The function known_edits2 removes words that are not in our original set and greatly reduces the size of edits2
 - E.g. all edit 2 combinations for “something” produces 114,324 words, but pruning leaves only 4 words: smoothing, seething, something, and soothing
- Now he assumes his probability model, which is “all known words of edit distance 1 are infinitely more probable than known words of edit distance 2
- He defines function correct which chooses as the set of candidate words the set with the shortest edit distance to the original word as long as the set has some known words
- See his web page for an evaluation of his program



Natural Language Corpus Data

- Peter Norvig has provided a web page that is full of useful data for a spelling corrector
 - <http://norvig.com/spell-correct.html>
- Peter Norvig's list of errors
- Peter Norvig's list of counts of single-edit errors
- <http://norvig.com/ngrams/>



Some References

- *How Difficult is it to Develop a Perfect Spell-checker? A Cross-linguistic Analysis through Complex Network Approach*, Monojit Choudhury¹, Markose Thomas², Animesh Mukherjee¹, Anupam Basu¹, and Niloy Ganguly¹

<http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=52A3B869596656C9DA285DCE83A0339F?doi=10.1.1.146.4390&rep=rep1&type=pdf>

- *Using the web for language independent spellchecking and autocorrection* by C. Whitelaw et al Proc. 2009 Conf. on Empirical Methods in Natural Language Processing, pp890-899

http://static.googleusercontent.com/external_content/untrusted_dlcp/research.google.com/en/us/pubs/archive/36180.pdf

Spell Checking by Computer, by Roger Mitton,

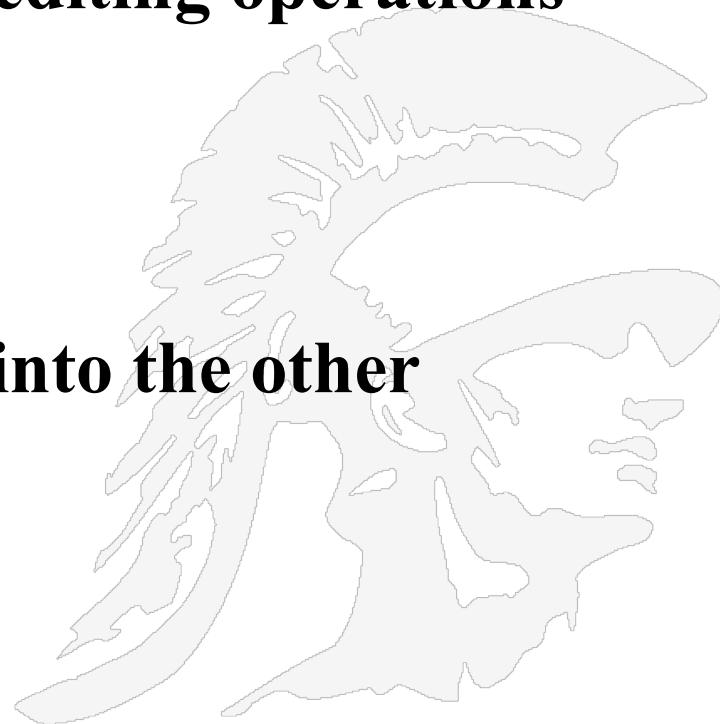
<http://www.dcs.bbk.ac.uk/~roger/spellchecking.html>

Edit Distance & Levenshtein Algorithm



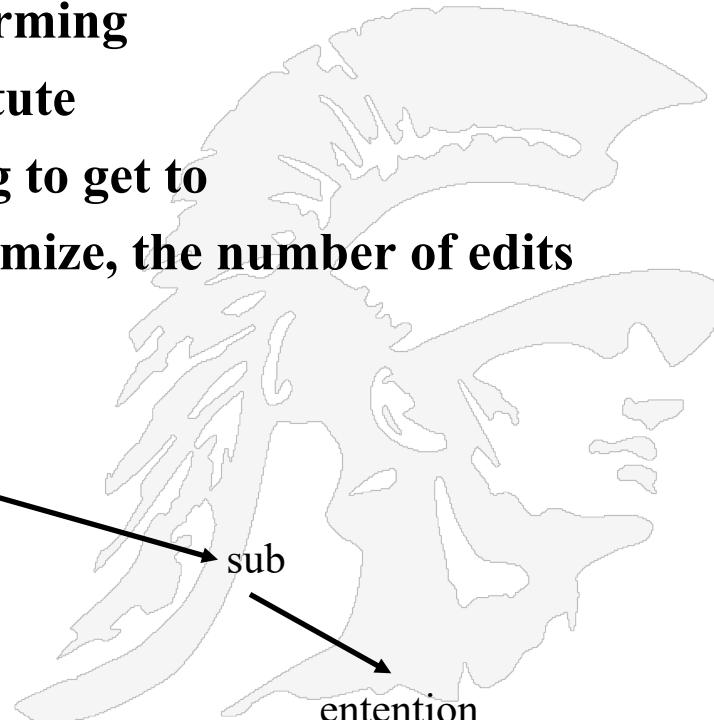
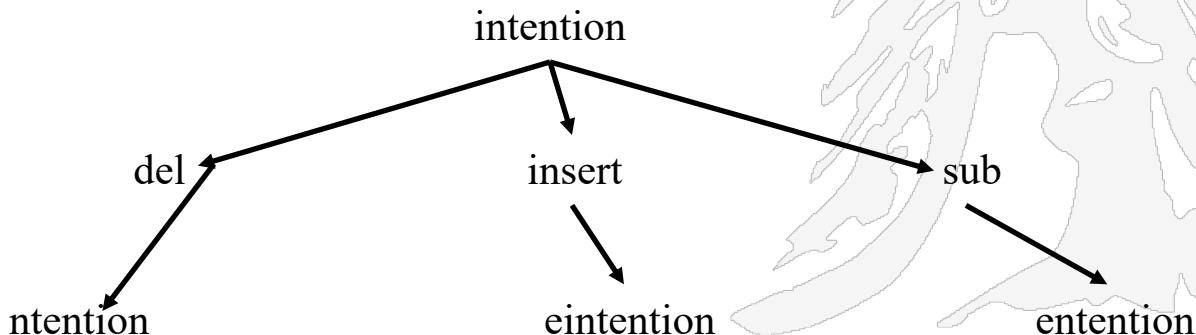
Edit Distance

- the minimum edit distance between two strings is the minimum number of editing operations
 - insertion
 - deletion
 - substitution
- needed to transform one into the other**



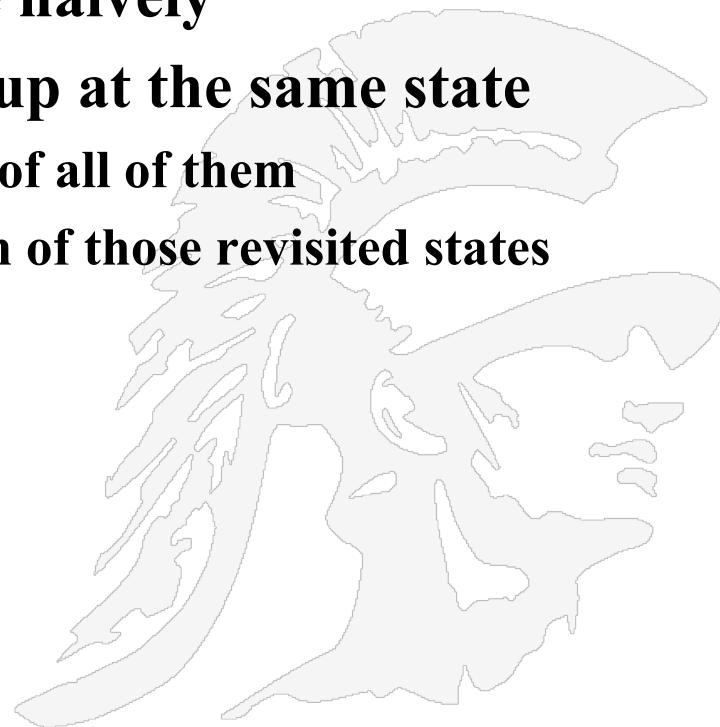
How to Find the Minimum Edit Distance

- searching for a path (sequence of edits) from the start string to the final string
 - initial state: word we're transforming
 - Operators: insert, delete, substitute
 - goal state: the word we're trying to get to
 - path cost: what we want to minimize, the number of edits



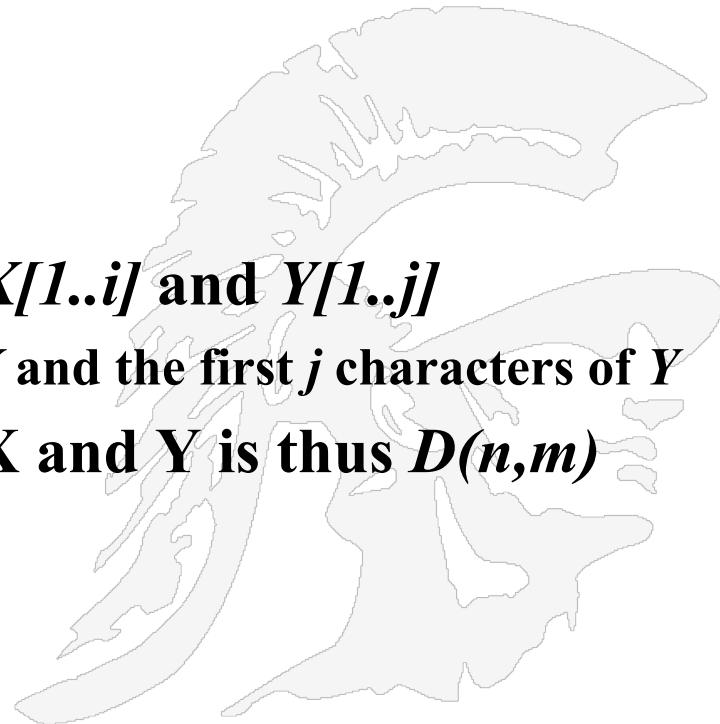
Minimum Edit as Search

- but the space of all edit sequences is huge!
 - we can't afford to navigate naively
 - lots of distinct paths wind up at the same state
 - we don't have to keep track of all of them
 - just the shortest path to each of those revisited states



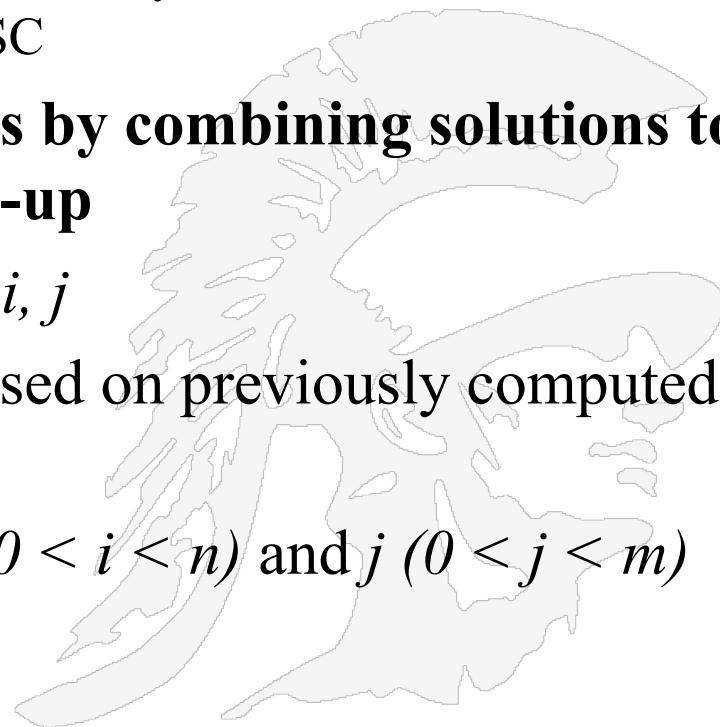
Define Min Edit Distance

- for two strings
 - X of length n
 - Y of length m
- we define $D(i, j)$ as
 - 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 $D(n,m)$



Dynamic Programming for Minimum Edit Distance

- **Dynamic programming: a tabular computation of $D(n,m)$**
 - by the way, the technique was invented by Richard Bellman while at the Rand Corp, later a professor at USC
- **Solving optimization problems by combining solutions to sub-optimal problems bottom-up**
 - we compute $D(i,j)$ for small i, j
 - and 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 ($0 < j < m$)





Defining Min Edit Distance (Levenshtein)

- Initialization

$$D(i, 0) = i$$

$$D(0, j) = j$$

- Recurrence Relation:

For each $i = 1 \dots M$

For each $j = 1 \dots N$

$$D(i, j) = \min \begin{cases} D(i-1, j) + 1 & \text{deletion} \\ D(i, j-1) + 1 & \text{insertion} \\ D(i-1, j-1) + 2; & \begin{cases} \text{if } X(i) \neq Y(j) \\ 0; & \text{if } X(i) = Y(j) \end{cases} \end{cases}$$

- Termination:

$D(N, M)$ is distance

cost 1 for insertion, 1 for deletion, and 2 for substitution;

initialization: for the first i characters of X , deleting each one to give us 0 characters is a cost of i

initialization: to create the first j characters of Y , we insert them for a cost of j

the distance in any cell is the minimum distance from one of three possible cells

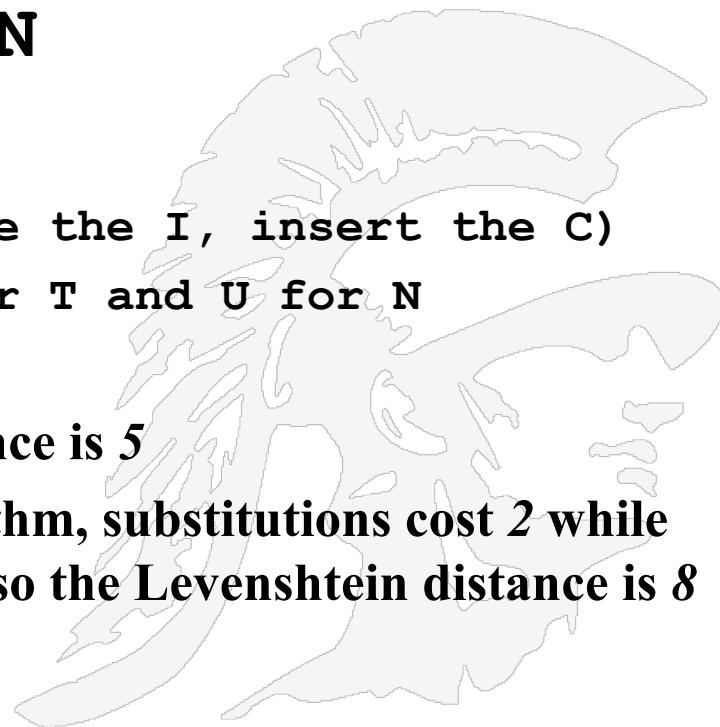
e.g. $D(i, j) = D(i-1, j)$ is deleting one character from X for a cost of 1

Minimum Edit Distance Example

- Given the two strings: intention and execution, what is their minimum edit distance?

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

- The * INDICATES A GAP (delete the I, insert the C)
- E is substituted for N, X for T and U for N
- TION line up successfully
- if each operation is 1, the edit distance is 5
- However, in the Levenshtein algorithm, substitutions cost 2 while deletions and insertions still cost 1, so the Levenshtein distance is 8



<https://class.coursera.org/nlp/lecture/7>

Dan Jurafsky



The Edit Distance Table

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

$D(i,j) = \min \begin{cases} D(i-1,j) + 1 \\ D(i,j-1) + 1 \\ D(i-1,j-1) + \begin{cases} 2; & \text{if } S_1(i) \neq S_2(j) \\ 0; & \text{if } S_1(i) = S_2(j) \end{cases} \end{cases}$

A red arrow points from the value 1 in the cell corresponding to 'I' and 'T' to the bottom row of the table.

Using the equation we can fill in the values in the table

is the empty string; to go from # to # costs 0;

to go from I to # is deleting the I for a cost of 1

to go from NI to # is deleting 2 characters for a cost of 2



Edit Distance

$$D(i,j) = \min \begin{cases} D(i,j-1) + 1 \\ D(i-1,j-1) + \begin{cases} 2; & \text{if } S_1(i) \neq S_2(j) \\ 0; & \text{if } S_1(i) = S_2(j) \end{cases} \end{cases}$$

N	9										
O	8										
I	7										
T	6										
N	5										
E	4										
T	3										
N	2	333									
I	1	2									
#	0	1	2	3	4	5	6	7	8	9	
	#	E	X	E	C	U	T	I	O	N	

to go from I to E that is either (deletion followed by insertion) or (substitution) both have a cost of 2
 to go from NI to E that is either (delete, delete, insert) or (delete, insert, delete), or (substitute, delete)
 all of which have a cost of 3

The Complete Edit Distance Table

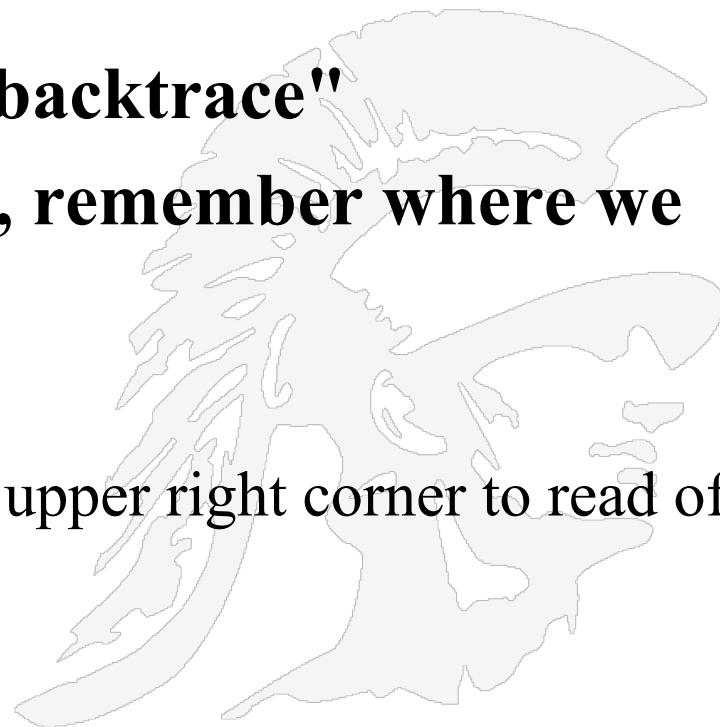
In the upper right corner the final edit distance for converting INTENTION to EXECUTION is 8 ; the edit distance for converting INTE to EXE IS 5

The screenshot shows a web browser window with the title "Computing Minimum Edit ...". The URL is <https://class.coursera.org/nlp/lecture/7>. The page features the Stanford University NLP logo and the heading "The Edit Distance Table". Below the heading is a 12x12 grid representing an edit distance matrix. The rows are labeled with the characters N, O, I, T, N, E, T, N, I, #, and an empty cell. The columns are labeled with the characters 9, 8, 9, 10, 11, 12, 11, 10, 9, 8, 9, and 8. The matrix contains various numbers representing edit distances, such as 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, and 10.

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

Computing Alignments

- **Edit distance isn't sufficient**
 - we often need to align each character of the two strings to each other
- **we do this by keeping a "backtrace"**
- **every time we enter a cell, remember where we came from**
- **when we reach the end,**
 - trace back the path from the upper right corner to read off the alignment



File Edit View History Bookmarks Tools Help

NLP Backtrace for Computing ... +

https://class.coursera.org/nlp/lecture/8

Search

Dan Jurafsky



Edit Distance

$$D(i,j) = \min \begin{cases} D(i-1,j) + 1 \\ D(i,j-1) + 1 \\ D(i-1,j-1) + \begin{cases} 2; & \text{if } S_1(i) \neq S_2(j) \\ 0; & \text{if } S_1(i) = S_2(j) \end{cases} \end{cases}$$

N	9									
O	8									
I	7									
T	6									
N	5									
E	4	3	4	5	4					
T	3	2	3	4	5					
N	2	1	2	3	4					
I	1	0	1	2	3					
#	0	1	2	3	4	5	6	7	8	9
#	E	X	E	C	U	T	I	O	N	

the same value in some cells can be arrived at in 3 different ways, but in other cells only in 2 different ways, or in other cells only in 1 way;
 e.g. how did we get from string I to string E: we could have come from any of the three possible paths however, the cheapest way to get from INTE and E is by 3 deletions for a value of 3; the other possibilities give a 4, so we reject them;

for every cell we have the path where the value could have come from

The screenshot shows a web browser window with the following details:

- Menu Bar:** File, Edit, View, History, Bookmarks, Tools, Help.
- Toolbar:** Includes icons for back, forward, search, and other common browser functions.
- Title Bar:** NLP Backtrace for Computing ...
- Address Bar:** https://class.coursera.org/nlp/lecture/8
- Content Area:** Displays the "MinEdit with Backtrace" interface. It features a grid of cells where each cell contains a character or a number. The grid is as follows:

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	9 ← 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	

The grid shows various paths (indicated by slashes and arrows) leading to each cell's value. A green dot is placed on the cell containing '9' in the second row, third column, indicating its origin. The Stanford University NLP logo is visible in the top left corner of the content area.

from the trace we see that N matches N, O matches O, I matches I, T matches T, but then we have insertion, etc.

we add pointers as we compute the min distance
 for insertion we point left
 for deletion we point down
 for substitution we point diagonally

Dan Jurafsky

Adding Backtrace to Minimum Edit Distance

- Base conditions:

$$D(i, 0) = i \quad D(0, j) = j \quad D(N, M) \text{ is distance}$$

- Recurrence Relation:

For each $i = 1 \dots M$

For each $j = 1 \dots N$

$$D(i, j) = \min \begin{cases} D(i-1, j) + 1 & \text{deletion} \\ D(i, j-1) + 1 & \text{insertion} \\ D(i-1, j-1) + 2; & \begin{cases} \text{if } X(i) \neq Y(j) & \text{substitution} \\ 0; & \text{if } X(i) = Y(j) \end{cases} \\ D(i-1, j-1) + 2; & \begin{cases} \text{if } X(i) \neq Y(j) & \text{substitution} \\ 0; & \text{if } X(i) = Y(j) \end{cases} \end{cases}$$

$$ptr(i, j) = \begin{cases} \text{LEFT} & \text{insertion} \\ \text{DOWN} & \text{deletion} \\ \text{DIAG} & \text{substitution} \end{cases}$$

Performance

- Time: $O(nm)$
- Space: $O(nm)$
- Backtrace: $O(n + m)$



Weighted Edit Distance

- why would we add weights to the computation?
 - spell correction: some letters are more likely to be mistyped than others
- a **confusion matrix** is a specific table layout that allows visualization of the performance of an algorithm; each column of the matrix represents the instances in a predicted class while each row represents the instances in an actual class (or vice-versa)

≡ 10/102 3 - 4 - Weighted Minimum Edit Distance - Stanford NLP - Professor Dan Jurafsky & Chris Manning



Dan Jurafsky



Confusion matrix for spelling errors

X	sub[X, Y] = Substitution of X (incorrect) for Y (correct)																									
	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	0	0	2	1	3	0	6	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	6	0	0	49	0	0	0	2	1	47	0	2	1	15	0	0
j	0	1	9	0	0	1	0	0	0	0	2	1	0	0	0	0	0	5	0	0	0	0	0	0	0	0
k	1	2	8	4	1	1	2	5	0	0	0	5	0	2	0	0	0	6	0	0	0	4	0	0	3	0
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	25	0	2	0	0	0	14	0	2	4	14	39	0	0	0	18	0	0	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	2	0	8	0	0
v	0	0	7	0	0	3	0	0	0	0	1	0	0	1	0	0	0	8	3	0	0	0	0	0	0	0
w	2	2	1	0	1	0	0	2	0	0	1	0	0	0	7	0	6	3	3	1	0	0	0	0	0	0
x	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	9	0	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	7	5	0	0	0	0	2	21	3	0	0	0	0	3	0	0	

the confusion matrix for spelling errors shows us, e.g. that "e" is most often confused with "a", and that "i" is often confused with both "e" and "a"

we can use the confusion matrix to add a special cost for insertions and deletions

The screenshot shows a web browser window with the following details:

- Title Bar:** File Edit View History Bookmarks Tools Help
NLP Weighted Minimum Edit D... +
- Address Bar:** https://class.coursera.org/nlp/lecture/9
- Content Area:**
 - Header:** Dan Jurafsky
 - Image:** Stanford University NLP logo
 - Title:** Weighted Min Edit Distance
 - List:**
 - Initialization:
 $D(0,0) = 0$
 $D(i,0) = D(i-1,0) + \text{del}[x(i)]; \quad 1 < i \leq N$
 $D(0,j) = D(0,j-1) + \text{ins}[y(j)]; \quad 1 < j \leq M$
 - Recurrence Relation:
$$D(i,j) = \min \begin{cases} D(i-1,j) + \text{del}[x(i)] \\ D(i,j-1) + \text{ins}[y(j)] \\ D(i-1,j-1) + \text{sub}[x(i),y(j)] \end{cases}$$
 - Termination:
 $D(N,M)$ is distance

del[] is a special deletion cost, etc