

Spell Checker

Leonardo Araújo

UFSJ



Spelling Correction

Spelling correction is an integral part of modern writing, ranging from **texting** and **emailing** to document creation and **web searches**. Despite their ubiquity, modern spell correctors aren't perfect, as evidenced by "autocorrect-gone-wrong" scenarios.

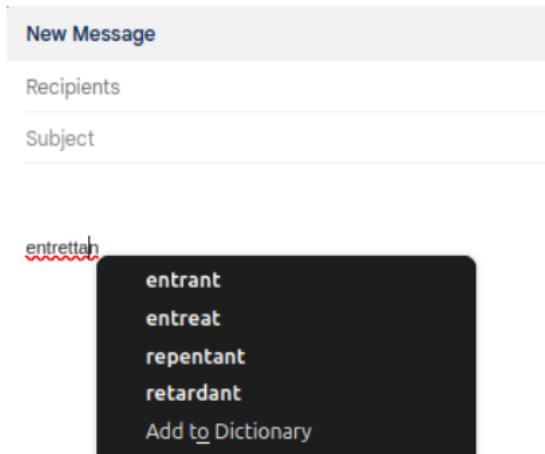


Figure 1: Spell checker.

Applications of Spell Checking

- Text Writing
- Automated and Information Systems
 - Data Entry Systems
 - Search and Information Retrieval
 - Optical Character Recognition (OCR)
 - Chatbots
 - Translation Systems

Automatic Spelling Correction Task

- 1 detection of an error;
- 2 generation of correction candidates;
- 3 ranking of candidate corrections;
- 4 perform automatic correction.

Perspectives in Spelling Correction

1. Non-Word Spelling Correction

- Detects and corrects errors where the **word does not exist** in the dictionary.

Example:

- Input: speling
- Correction: spelling

2. Real-Word Spelling Correction

- Detects and corrects errors where the **word exists** but is contextually wrong.

Example:

- Input: I no what to do.
- Correction: I know what to do.

Error Sources in Spelling

1 Typographical Errors:

- May change with input devices (physical or virtual keyboard, or OCR system) and environment conditions.
- Insertion: `speeling` → `spelling`
- Deletion: `spelng` → `spelling`
- Substitution: `spolling` → `spelling`
- Transposition: `spelilng` → `spelling`
- Diacritical marking: `naive` → `naïve`

2 Homophone Errors:

- Homophones: `their` / `there`
- Near-homophones: `accept` / `except`

3 Grammatical Errors:

- `among` / `between`

4 Cross Word Boundary Errors:

- `maybe` / `may be`

Notable Algorithms and Tools

- **Soundex** (1918): Phonetic algorithm that maps similar-sounding names.

Stephen → S315, Perez → P620, Juice → J200, Robert → R163

Steven → S315, Powers → P620, Juicy → J200, Rupert → R163

Stefan → S315, Price → P620, Juiced → J230, Rubin → R150

■ **Shannon (1948):** A Mathematical Theory of Communication.

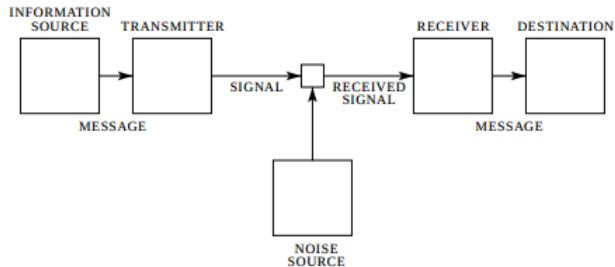


Fig. 1—Schematic diagram of a general communication system.

Figure 2: Noisy Channel.

- **Shannon (1950):** Introduction of n-gram models in text analysis.



Figure 3: Word prediction is mean?

- **Blair (1960):** Early algorithm for spelling error correction.
 - Blair introduced the concept of similarity keys to group words based on their likelihood of being confused with one another.
 - r-letter abbreviation of an n-letter word
 - Information conveyed is inversely proportional to its a priori probability of occurrence.
 - eliminate $n - r$ letters in the order of their expected frequency
 - eliminate by their frequency of their occurrence as errors (best approach)
 - weight must also be given to the position of the letter in the word

EXAMPLE

A B S O R B E N T		A B S O R B A N T
5 1 5 4 4 1 7 3 3.	Letter score	5 1 5 4 4 1 5 3 3
0 2 4 5 5 5 4 3 1	Position score	0 2 4 5 5 5 4 3 1
5 3 9 9 9 6 11 6 4	Sum of scores	5 3 9 9 9 6 9 6 4
* * * * *	Delete	* * * * *
A B B T	Abbreviation	A B B T

Figure 4: Example of Blair's algorithm.

- **Damerau–Levenshtein** distance (1964, 1966): A string metric for measuring the edit distance between two sequences.

		S	u	n	d	a	y
	0	1	2	3	4	5	6
S	1	0	1	2	3	4	5
a	2	1	1	2	3	3	4
t	3	2	2	2	3	4	4
u	4	3	2	3	3	4	5
r	5	4	3	3	4	4	5
d	6	5	4	4	3	4	5
a	7	6	5	5	4	3	4
y	8	7	6	6	5	4	3

Figure 5: How many operations does it take to turn Saturday into Sundat?

The Levenshtein distance between two strings a, b (of length $|a|$ and $|b|$ respectively) is given by

$$\text{lev}(a, b) = \begin{cases} |a| & \text{if } |b| = 0, \\ |b| & \text{if } |a| = 0, \\ \text{lev}(\text{tail}(a), \text{tail}(b)) & \text{if } \text{head}(a) = \text{head}(b), \\ 1 + \min \begin{cases} \text{lev}(\text{tail}(a), b) & \text{deletion} \\ \text{lev}(a, \text{tail}(b)) & \text{insertion} \\ \text{lev}(\text{tail}(a), \text{tail}(b)) & \text{replacement} \end{cases} & \text{otherwise} \end{cases}$$

Damerau-Levenshtein distance: also allows transposition of adjacent symbols.

Operations are expensive and language dependent: e.g. as of version 16.0, Unicode defines a total of 98682 Chinese characters.

- **BK-Trees** (1973): Efficient search for near matches using Levenshtein distance.
 - An arbitrary element a is selected as root node.
 - The k -th subtree is recursively built of all elements b such that $d(a, b) = k$.
 - Search idea: restrict the exploration of the tree to nodes that can only improve the best candidate found so far (use triangle inequality).

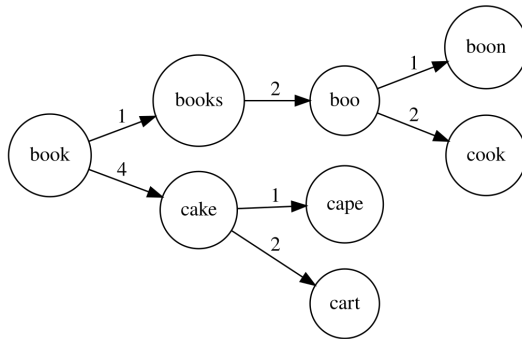
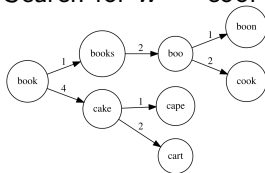


Figure 6: Burkhard-Keller Tree.

Search for $w = \text{'cool'}$



- 1 $d_u = d(w, u) = d(\text{'cool'}, \text{'book'}) = 2$, set $d_{\text{best}} = 2$;
- 2 $v = \text{'books'}$, $|d_{uv} - d_u| = |1 - 2| = 1 < d_{\text{best}}$, then select v ;
- 3 $v = \text{'cake'}$, $|d_{uv} - d_u| = |4 - 2| = 2 \not< d_{\text{best}}$, do not select v ;
- 4 $d_u = d(w, u) = d(\text{'cool'}, \text{'books'}) = 3$, $d_u \not< d_{\text{best}}$;
- 5 $d_u = d(w, u) = d(\text{'cool'}, \text{'boo'}) = 2$, $d_u \not< d_{\text{best}}$;
- 6 $v = \text{'boon'}$, $|d_{uv} - d_u| = |2 - 1| = 1 < d_{\text{best}}$, then select v ;
- 7 $v = \text{'cook'}$, $|d_{uv} - d_u| = |2 - 2| = 0 < d_{\text{best}}$, then select v ;
- 8 $d_u = d(w, u) = d(\text{'cool'}, \text{'cook'}) = 1$, $d_u < d_{\text{best}}$, set $d_{\text{best}} = 1$;
- 9 $d_u = d(w, u) = d(\text{'cool'}, \text{'boon'}) = 2$, $d_u \not< d_{\text{best}}$;
- 10 'cook' is returned as the answer with $d_{\text{best}} = 1$.

■ SPELL (Unix, 1975)

- Error detection only.
- Prefix and suffix removal (reduces the list below 1/3);
 - buzzed → buzz, mapping → map, possibly → possible, antisocial → social, metaphysics → physics.
- Hashing (discarding 60% of the remaining bits);
Examples of hashing functions:
 - 1 Shift-and-Add: $h = (h \ll 1) + \text{char} \% m$
 - 2 Multiplicative Hashing: $h = (a \cdot h + \text{char}) \% m$ (with a typically 31 or 33)
 - 3 XOR-based Hashing: $h = h \oplus (\text{char} \ll k)$
- Words were represented by 16-bit machine words;
- Bloom filter;
- False Positives.

■ Jaro similarity (1989)

The Jaro similarity sim_j of two given strings s_1 and s_2 is

$$sim_j = \begin{cases} 0 & \text{if } m = 0 \\ \frac{1}{3} \left(\frac{m}{|s_1|} + \frac{m}{|s_2|} + \frac{m-t}{m} \right) & \text{otherwise} \end{cases}$$

where:

- $|s_i|$ is the length of the string s_i ;
- m is the number of "matching characters" (see below);
- t is the number of "transpositions" (see below).

Jaro similarity score is 0 if the strings do not match at all, and 1 if they are an exact match. In the first step, each character of s_1 is compared with all its matching characters in s_2 . Two characters from s_1 and s_2 respectively, are considered **matching** only if they are the same and not farther than $\left\lfloor \frac{\max(|s_1|, |s_2|)}{2} \right\rfloor - 1$ characters apart. **Transposition** is the number of matching characters that are not in the right order divided by two.

■ Jaro-Winkler similarity (1990)

- Introduces Winkler modification.
- Prefix Length ℓ : if two strings share a common prefix, they are likely to be more similar.
- scale factor p : enhances the Jaro similarity score based on the length of the common prefix (usually set to 0.1 and should not exceed 0.25).

$$sim_w = sim_j + \ell p(1 - sim_j)$$

- **Metaphone** (1990), Double Metaphone (2000), Metaphone 3 (2009): Extracts phonetic information for better matching.
 - Set of rules to improve on the Soundex algorithm.
 - Smith → SMO, [SMO, XMT], Schmidt → SXMTT, [XMT, SMT],
 - Taylor → TLR, [TLR], Taylor → EFNS, [AFNS],
 - Roberts → RBRTS, [RPRTS]
 - spelling → SPLNK, [SPLNK], speling → SPLNK, [SPLNK], speeling → SPLNK, [SPLNK], sprlling → SPRLNK, [SPRLNK]

- **Noisy Channel Model (Kernighan et al., 1990 and Mays et al., 1991):**
Combined prior and likelihood models.

In the noisy channel model, we imagine that the surface form we see is actually a “distorted” form of an original word passed through a noisy channel. The decoder passes each hypothesis through a model of this channel and picks the word that best matches the surface noisy word. (Jurafsky and Martin, 2024)

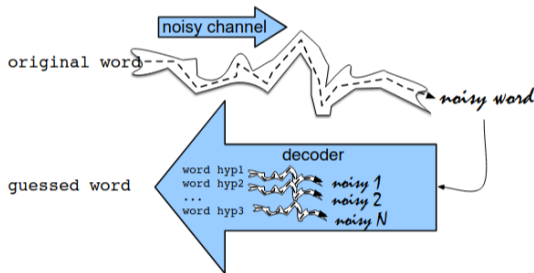


Figure 7: Noisy Channel Model

This noisy channel model is a kind of **Bayesian inference**.

Out of all possible words in the vocabulary V we want to find the word w such that $P(w|x)$ is highest.

$$\hat{w} = \arg \max_{w \in V} P(w|x)$$

Using Bayes: $P(x, w) = P(w|x)P(x) = P(x|w)P(w)$,

$$\hat{w} = \arg \max_{w \in V} \frac{P(x|w)P(w)}{P(x)} = \arg \max_{w \in V} \underbrace{P(x | w)}_{\text{channel model or likelihood}} \underbrace{P(w)}_{\text{prior}}$$

$$\hat{w} = \arg \max_{w \in V} (\log P(x | w) + \log P(w))$$

```

function NOISY CHANNEL SPELLING(word  $x$ , dict  $D$ ,  $lm$ , editprob) returns correction

  if  $x \notin D$ 
    candidates, edits  $\leftarrow$  All strings at edit distance 1 from  $x$  that are  $\in D$ , and their edit
    for each  $c, e$  in candidates, edits
      channel  $\leftarrow$  editprob( $e$ )
      prior  $\leftarrow lm(c)$ 
       $score[c] = \log channel + \log prior$ 
    return  $\operatorname{argmax}_c score[c]$ 

```

Figure 8: Noisy channel model for spelling correction for unknown words (Jurafsky and Martin, 2024).

Example

original word

actress

cress

caress

access

across

acres

?

noisy channel



acress

Figure 9: Example: misspelling across.

Transformation					
Error	Correction	Correct Letter	Error Letter	Position (Letter #)	Type
acress	actress	t	—	2	deletion
acress	cress	—	a	0	insertion
acress	caress	ca	ac	0	transposition
acress	access	c	r	2	substitution
acress	across	o	e	3	substitution
acress	acres	—	s	5	insertion
acress	acres	—	s	4	insertion

Figure 10: Candidate corrections for the misspelling acress and the transformations that would have produced the error (after Kernighan et al. (1990)). “—” represents a null letter. (Jurafsky and Martin, 2024)

w	count(w)	p(w)
actress	9,321	.0000231
cress	220	.000000544
caress	686	.00000170
access	37,038	.0000916
across	120,844	.000299
acres	12,874	.0000318

Figure 11: Language model from the 404,253,213 words in the Corpus of Contemporary English (COCA) (Jurafsky and Martin, 2024).

Error model

- A perfect model would need all sorts of factors: who the typist was, whether the typist was left-handed or right-handed, and so on.
- We can get a pretty reasonable estimate of $P(x|w)$ just by looking at **local context**: the identity of the correct letter itself, the misspelling, and the surrounding letters.
- Confusion Matrices:
 - $\text{del}[x, y]$: count(xy typed as x)
 - $\text{ins}[x, y]$: count(x typed as xy)
 - $\text{sub}[x, y]$: count(x typed as y)
 - $\text{trans}[x, y]$: count(xy typed as yx)

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

Figure 12: Confusion matrix for spelling errors (Kernighan et al., 1990).

Estimating the channel model

$$P(x|w) = \begin{cases} \frac{\text{del}[x_{i-1}, w_i]}{\text{count}[x_{i-1} w_i]}, & \text{if deletion} \\ \frac{\text{ins}[x_{i-1}, w_i]}{\text{count}[w_{i-1}]}, & \text{if insertion} \\ \frac{\text{sub}[x_i, w_i]}{\text{count}[w_i]}, & \text{if substitution} \\ \frac{\text{trans}[w_i, w_{i+1}]}{\text{count}[w_i w_{i+1}]}, & \text{if transposition} \end{cases}$$

Candidate Correction	Correct Letter	Error Letter	$x w$	$P(x w)$
actress	t	-	c ct	.000117
cress	-	a	a #	.00000144
caress	ca	ac	ac ca	.00000164
access	c	r	r c	.000000209
across	o	e	e o	.0000093
acres	-	s	es e	.0000321
acres	-	s	ss s	.0000342

Figure 13: Channel model for acres; the probabilities are taken from the `del[]`, `ins[]`, `sub[]`, and `trans[]` confusion matrices as shown in Kernighan et al. (1990).

Final probabilities for each of the potential corrections

Candidate	Correct	Error				
Correction	Letter	Letter	x w	P(x w)	P(w)	$10^9 * P(x w)P(w)$
actress	t	-	c ct	.000117	.0000231	2.7
cress	-	a	a #	.00000144	.000000544	0.00078
caress	ca	ac	ac ca	.00000164	.00000170	0.0028
access	c	r	r c	.000000209	.0000916	0.019
across	o	e	e o	.0000093	.000299	2.8
acres	-	s	es e	.0000321	.0000318	1.0
acres	-	s	ss s	.0000342	.0000318	1.0

Figure 14: Computation of the ranking for each candidate correction, using the language model shown earlier and the error model. The final score is multiplied by 10^9 for readability (Jurafsky and Martin, 2024).

*Unfortunately, the algorithm was wrong here; the writer's intention becomes clear from the context: ... was called a "stellar and versatile **acress** whose combination of sass and glamour has defined her ...". The surrounding words make it clear that actress and not across was the intended word. (Jurafsky and Martin, 2024)*

Using the *Corpus of Contemporary American English* to compute bigram probabilities for the words *actress* and *across* in their context using add-one smoothing, we get the following probabilities:

$$P(\text{actress}|\text{versatile}) = .000021$$

$$P(\text{across}|\text{versatile}) = .000021$$

$$P(\text{whose}|\text{actress}) = .0010$$

$$P(\text{whose}|\text{across}) = .000006$$

Multiplying these out gives us the language model estimate for the two candidates in context:

$$P(\text{versatile actress whose}) = .000021 \times .0010 = 210 \times 10^{-10}$$

$$P(\text{versatile across whose}) = .000021 \times .000006 = 1 \times 10^{-10}$$

Jurafsky, D., & Martin, J. H. (2024). *Speech and Language Processing*.

Kernighan, M. D. et al. (1990). *A spelling correction program based on a noisy channel model*.

Mays, E. et al. (1991). *Context based spelling correction*.

- Noisy Channel Model
 - Correct (Unix, 1990): Takes inputs from SPELL rejected words and provides candidates. Operations: Insertion, Deletion, Substitution, Reversal. Uses error probabilities.

- **Brill-Moore channel model (2000):** String to string edits.
 - Let Σ be an alphabet, the model allows all edit operations of the form $\alpha \rightarrow \beta$, where $\alpha, \beta \in \Sigma^*$.
 - $P(\alpha \rightarrow \beta)$ is the probability that when users intends to type α and they typed β instead.
 - $P(\alpha \rightarrow \beta | PNS)$ probability conditioned by the position on the string
 - $P(e | a)$ does not vary greatly with position.
 - $P(\text{ent} | \text{ant})$ is highly dependent upon position.
 - People rarely mistype *antler* as *entler*, but often mistype *reluctant* as *reluctent*.

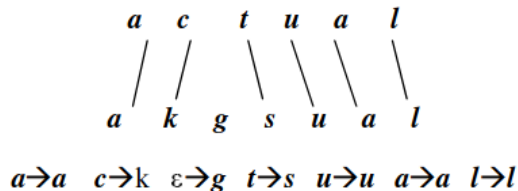


Figure 15: String alignment (Brill and Moore, 2000).

Brill, E. and Moore, R. C. (2000). *An Improved Error Model for Noisy Channel Spelling Correction*.

- **Aspell** (2000): Combines spelling and phonetic correction.
 - Hashing for Spell Checking: Efficient candidate lookup using hash tables.
 - Metaphone Algorithm: Handles phonetic corrections by matching words that sound similar.
 - Ispell's Near Miss Strategy:
 - Focuses on edit distance 1 to reduce the search space.
 - Early Dictionary Filtering: Prunes invalid candidates during generation.

Example - Handling Homophones in Aspell

- Misspelled word ther
- Candidates: there, their, they're

1 Metaphone

- The Metaphone algorithm transforms words into phonetic codes based on pronunciation.
- Phonetic codes for the candidate words:
 - there → OR
 - their → OR
 - they're → OR
- Homophones share the same code (OR).

2 Workflow:

- Input: Misspelled word ther.
- Step 1: Generate candidates using **edit distance 2 or less**:
 - Candidates: there, their, thee, thor, her, the, they're.
- Step 2: Compute Metaphone codes for all candidates:
 - Candidates phonetically similar to ther (OR) rank higher: there, their, thor, they're.
- Step 3: Rank and suggest based on:
 - Word frequency, Edit distance, Phonetic Similarity, Error Likelihood.

3 Limitations:

- Metaphone matches words by sound but lacks **contextual understanding**.
- Example:
 - Input: *"I went to ther house."*
 - Suggestions: thee, their, there, therm, the, her, Thar, Thea, Thor, Thur.
 - Aspell cannot infer the correct word (their) without considering the sentence's context.

- **Hunspell** (2002): Morphological analyzer with affix rules and phonetic matching.

Key Features:

- **Morphological Analysis:**
 - Supports complex languages with rich morphology (e.g., Hungarian, Turkish, Finnish).
 - Handles word roots, prefixes, and suffixes using affix rules.
- **Dictionary System:**
 - Two components:
 - 1 Dictionary File: Contains root forms of words.
 - 2 Affix File: Defines rules for combining roots with prefixes/suffixes.
- **Levenshtein Distance:**
 - Uses *edit distance* to generate and rank candidate corrections.
- **Phonetic Matching:**
 - Uses a table-driven phonetic transcription algorithm borrowed from Aspell. It is useful for languages with not pronunciation based orthography.
- **n-gram similarity:**
 - Improve suggestions.

- Multilingual Support:
 - Available for 98 languages with extensive dictionaries.

Applications:

- Integrated into tools like LibreOffice, Firefox, and Chrome for multilingual spell checking.
- Supports custom dictionaries for specialized fields (e.g., medical, legal).

[Hunspell at GitHub](#)

- **Norvig's Algorithm (2007):** Uses Damerau-Levenshtein distance to generate candidates.

Key Features:

- **Edit Distance:**
 - Generates all possible words within a given edit distance (e.g., 1 or 2) from the misspelled word.
 - Handles insertion, deletion, substitution, and transposition.
- **Dictionary Lookup:**
 - Filters candidates by validating them against a word dictionary.
- **Ranking:**
 - Ranks valid candidates based on:
 - **Word Frequency:** More frequent words are prioritized.
 - **Likelihood of Errors:** Based on the Noisy Channel Model (optional).

How to Write a Spelling Corrector

- **QWERTY Weighted Levenshtein Distance:** takes keyboard distance into account.

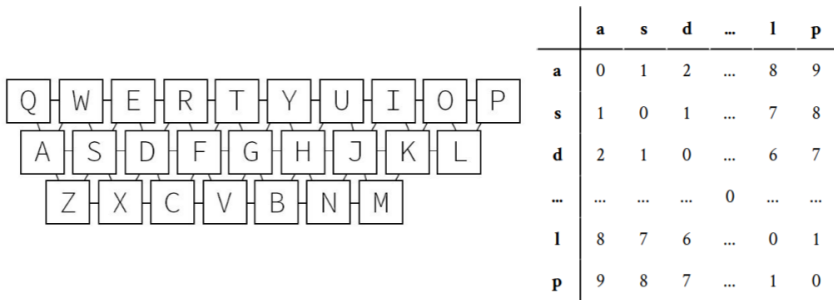


Figure 16: QWERTY keyboard and keyboard distance matrix.

- Distance between keys are in $[0,9]$. They are multiplied by $2/9$.

- Deletion: weighted by the average of the distances to the adjacent characters in the string.
- Insertion: unchanged, weight 1.
- Substitution: weighted according to the distance between the character that is removed and the character that is inserted.
- Transposition: unchanged, weight 1.

Samuelsson, 2017

- **Neural-Based Models:** Leverage deep learning for advanced error detection and correction.
 - Utilize deep learning techniques to improve spellchecking:
 - Recurrent Neural Networks (RNNs)
 - Word Embeddings
 - Transformers
 - Contextual Awareness
 - Learning from Data
 - Handling Typos

Examples:

- Google's Smart Compose
- Grammarly
- Microsoft Editor
- LanguageTool

Lexical Similarity Metrics

1 **Levenshtein Distance** (Edit Distance):

Minimal number of insertions, deletions, and replacements to transform one word into another.

2 **Jaro Similarity:**

Measures similarity based on matching characters and transpositions.

3 **Keyboard Distance:**

Considers physical proximity of keys.

4 **Phonetic Matching:**

Algorithms like Soundex and Metaphone to identify similar-sounding words.

Domain-Specific Spell Checkers

1 Medical

- MedSpell: a medical spelling and autocorrect application
- OpenMedSpel (open-source)

2 Programming

- CodeSpell: designed primarily for checking misspelled words in source code

3 Learning

- Kidspell: A child-oriented, rule-based, phonetic spellchecker

4 Accessibility

- Real Check: A Spellchecker for Dyslexia

5 Custom Dictionaries

- Hunspell and Aspell: Add specialized vocabularies

References

- Shannon, C. E. (1950). *Prediction and Entropy of Printed English*.
- Blair, C. R. (1960). *A program for correcting spelling errors*.
- Damerau, F. J. (1964). *A technique for computer detection and correction of spelling errors*.
- Levenshtein, V. I. (1966). *Binary codes capable of correcting deletions, insertions, and reversals*.
- Burkhard W. and Keller R. (1973). *Some approaches to best-match file searching*.
- Jaro, M. A. (1989). *Advances in Record-Linkage Methodology as Applied to Matching the 1985 Census of Tampa, Florida*.
- Winkler, W. E. (1990). *String Comparator Metrics and Enhanced Decision Rules in the Fellegi-Sunter Model of Record Linkage*.
- Kernighan, M. D. et al. (1990). *A spelling correction program based on a noisy channel model*.
- Mays, E. et al. (1991). *Context based spelling correction*.
- Atkinson, K. (2000). *GNU Aspell*.
- Németh, L. (2002). *Hunspell*.
- Samuelsson, A. (2017). *Weighting Edit Distance to Improve Spelling Correction in Music Entity Search*.
- Brill, E. and Moore, R. C. (2000). *An Improved Error Model for Noisy Channel Spelling Correction*.
- Jurafsky, D., & Martin, J. H. (2024). *Speech and Language Processing*.