CHAPTER

5 Spelling Correction and the Noisy Channel

ALGERNON: *But my own sweet Cecily, I have never written you any letters.*
CECILY: *You need hardly remind me of that, Ernest. I remember only too well that I was forced to write your letters for you. I wrote always three times a week, and sometimes oftener.*
ALGERNON: *Oh, do let me read them, Cecily?*
CECILY: *Oh, I couldn't possibly. They would make you far too conceited. The three you wrote me after I had broken off the engagement are so beautiful, and so badly spelled, that even now I can hardly read them without crying a little.*
Oscar Wilde, *The Importance of Being Earnest*

Like Oscar Wilde's fabulous Cecily, a lot of people were thinking about spelling during the last turn of the century. Gilbert and Sullivan provide many examples. *The Gondoliers*' Giuseppe, for example, worries that his private secretary is "shaky in his spelling", while *Iolanthe*'s Phyllis can "spell every word that she uses". Thorstein Veblen's explanation (in his 1899 classic *The Theory of the Leisure Class*) was that a main purpose of the "archaic, cumbrous, and ineffective" English spelling system was to be difficult enough to provide a test of membership in the leisure class.

Whatever the social role of spelling, we can certainly agree that many more of us are like Cecily than like Phyllis. Estimates for the frequency of spelling errors in human-typed text vary from 1-2% for carefully retyping already printed text to 10-15% for web queries.

In this chapter we introduce the problem of detecting and correcting spelling errors. Fixing spelling errors is an integral part of writing in the modern world, whether this writing is part of texting on a phone, sending email, writing longer documents, or finding information on the web. Modern spell correctors aren't perfect (indeed, autocorrect-gone-wrong is a popular source of amusement on the web) but they are ubiquitous in pretty much any software that relies on keyboard input.

Spelling correction is often considered from two perspectives. **Non-word spelling correction** is the detection and correction of spelling errors that result in non-words (like *graffe* for *giraffe*). By contrast, **real word spelling correction** is the task of detecting and correcting spelling errors even if they accidentally result in an actual word of English (**real-word errors**). This can happen from typographical errors (insertion, deletion, transposition) that accidentally produce a real word (e.g., *there* for *three*), or **cognitive errors** where the writer substituted the wrong spelling of a homophone or near-homophone (e.g., *dessert* for *desert*, or *piece* for *peace*).

real-word errors

Non-word errors are detected by looking for any word not found in a dictionary. For example, the misspelling *graffe* above would not occur in a dictionary. The larger the dictionary the better; modern systems often use enormous dictionaries derived from the web. To correct non-word spelling errors we first generate

candidates

candidates: real words that have a similar letter sequence to the error. Candidate corrections from the spelling error *graffe* might include *giraffe*, *graf*, *gaffe*, *grail*, or *craft*. We then rank the candidates using a **distance metric** between the source and the surface error. We'd like a metric that shares our intuition that *giraffe* is a more likely source than *grail* for *graffe* because *giraffe* is closer in spelling to *graffe* than *grail* is to *graffe*. The minimum edit distance algorithm from Chapter 2 will play a role here. But we'd also like to prefer corrections that are more frequent words, or more likely to occur in the context of the error. The noisy channel model introduced in the next section offers a way to formalize this intuition.

Real word spelling error detection is a much more difficult task, since any word in the input text could be an error. Still, it is possible to use the noisy channel to find candidates for each word w typed by the user, and rank the correction that is most likely to have been the users original intention.

5.1 The Noisy Channel Model

In this section we introduce the noisy channel model and show how to apply it to the task of detecting and correcting spelling errors. The noisy channel model was applied to the spelling correction task at about the same time by researchers at AT&T Bell Laboratories (Kernighan et al. 1990, Church and Gale 1991) and IBM Watson Research (Mays et al., 1991).

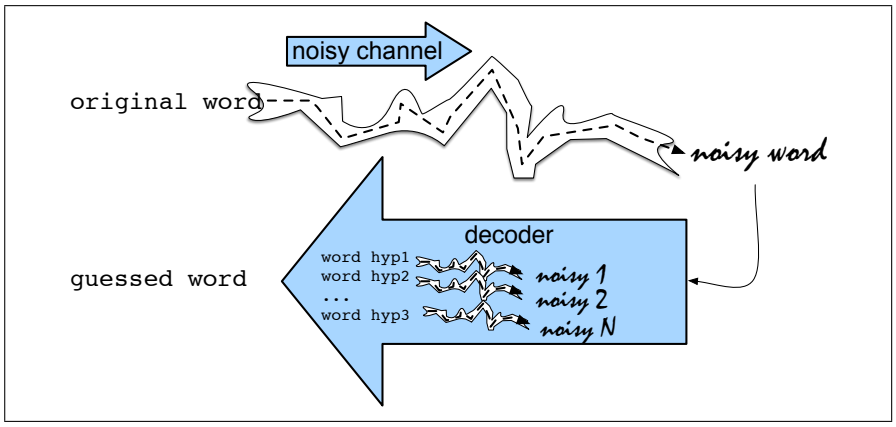


Figure 5.1 In the noisy channel model, we imagine that the surface form we see is actually a "distorted" form of an original word passed through a noisy channel. The decoder passes each hypothesis through a model of this channel and picks the word that best matches the surface noisy word.

noisy channel

The intuition of the **noisy channel** model (see Fig. 5.1) is to treat the misspelled word as if a correctly spelled word had been "distorted" by being passed through a noisy communication channel.

This channel introduces "noise" in the form of substitutions or other changes to the letters, making it hard to recognize the "true" word. Our goal, then, is to build a model of the channel. Given this model, we then find the true word by passing every word of the language through our model of the noisy channel and seeing which one comes the closest to the misspelled word.

Bayesian

This noisy channel model is a kind of **Bayesian inference**. We see an obser-

vation x (a misspelled word) and our job is to find the 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 ˆ to mean "our estimate of the correct word".

$$\hat{w} = \operatorname*{argmax}_{w \in V} P(w|x) \tag{5.1}$$

argmax

The function $\textbf{argmax}_x f(x)$ means "the x such that $f(x)$ is maximized". Equation 5.1 thus means, that out of all words in the vocabulary, we want the particular word that maximizes the right-hand side $P(w|x)$.

The intuition of Bayesian classification is to use Bayes' rule to transform Eq. 5.1 into a set of other probabilities. Bayes' rule is presented in Eq. 5.2; it gives us a way to break down any conditional probability $P(a|b)$ into three other probabilities:

$$P(a|b) = \frac{P(b|a)P(a)}{P(b)} \tag{5.2}$$

We can then substitute Eq. 5.2 into Eq. 5.1 to get Eq. 5.3:

$$\hat{w} = \operatorname*{argmax}_{w \in V} \frac{P(x|w)P(w)}{P(x)} \tag{5.3}$$

We can conveniently simplify Eq. 5.3 by dropping the denominator $P(x)$. Why is that? Since we are choosing a potential correction word out of all words, we will be computing $\frac{P(x|w)P(w)}{P(x)}$ for each word. But $P(x)$ doesn't change for each word; we are always asking about the most likely word for the same observed error x, which must have the same probability $P(x)$. Thus, we can choose the word that maximizes this simpler formula:

$$\hat{w} = \operatorname*{argmax}_{w \in V} P(x|w)\,P(w) \tag{5.4}$$

To summarize, the noisy channel model says that we have some true underlying word w, and we have a noisy channel that modifies the word into some possible misspelled observed surface form. The **likelihood** or **channel model** of the noisy channel producing any particular observation sequence x is modeled by $P(x|w)$. The **prior probability** of a hidden word is modeled by $P(w)$. We can compute the most probable word $\hat{w}$ given that we've seen some observed misspelling x by multiplying the prior $P(w)$ and the likelihood $P(x|w)$ and choosing the word for which this product is greatest.

likelihood
channel model
prior probability

We apply the noisy channel approach to correcting non-word spelling errors by taking any word not in our spell dictionary, generating a list of **candidate words**, ranking them according to Eq. 5.4, and picking the highest-ranked one. We can modify Eq. 5.4 to refer to this list of candidate words instead of the full vocabulary V as follows:

$$\hat{w} = \operatorname*{argmax}_{w \in C} \overbrace{P(x|w)}^{\text{channel model}} \overbrace{P(w)}^{\text{prior}} \tag{5.5}$$

The noisy channel algorithm is shown in Fig. 5.2.

To see the details of the computation of the likelihood and the prior (language model), let's walk through an example, applying the algorithm to the example misspelling *acress*. The first stage of the algorithm proposes candidate corrections by

```
function NOISY CHANNEL SPELLING(word x, dict D, lm, editprob) returns correction
  if x ∉ D
    candidates, edits ← All strings at edit distance 1 from x that are ∈ D, and their edit
    for each c, e in candidates, edits
      channel ← editprob(e)
      prior ← lm(x)
      score[c] = log channel + log prior
    return argmax_c score[c]
```

Figure 5.2 Noisy channel model for spelling correction for unknown words.

finding words that have a similar spelling to the input word. Analysis of spelling error data has shown that the majority of spelling errors consist of a single-letter change and so we often make the simplifying assumption that these candidates have an edit distance of 1 from the error word. To find this list of candidates we'll use the minimum edit distance algorithm introduced in Chapter 2, but extended so that in addition to insertions, deletions, and substitutions, we'll add a fourth type of edit, transpositions, in which two letters are swapped. The version of edit distance with transposition is called **Damerau-Levenshtein** edit distance. Applying all such single transformations to *acress* yields the list of candidate words in Fig. 5.3.

Damerau-Levenshtein

		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 5.3 Candidate corrections for the misspelling *acress* and the transformations that would have produced the error (after Kernighan et al. (1990)). "—" represents a null letter.

Once we have a set of a candidates, to score each one using Eq. 5.5 requires that we compute the prior and the channel model.

The prior probability of each correction $P(w)$ is the language model probability of the word w in context, which can be computed using any language model, from unigram to trigram or 4-gram. For this example let's start in the following table by assuming a unigram language model. We computed the language model from the 404,253,213 words in the Corpus of Contemporary English (COCA).

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

channel model

How can we estimate the likelihood $P(x|w)$, also called the **channel model** or

error model

error model? A perfect model of the probability that a word will be mistyped would condition on all sorts of factors: who the typist was, whether the typist was left-handed or right-handed, and so on. Luckily, 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. For example, the letters *m* and *n* are often substituted for each other; this is partly a fact about their identity (these two letters are pronounced similarly and they are next to each other on the keyboard) and partly a fact about context (because they are pronounced similarly and they occur in similar contexts).

A simple model might estimate, for example, $p(acress|across)$ just using the number of times that the letter *e* was substituted for the letter *o* in some large corpus of errors. To compute the probability for each edit in this way we'll need a **confusion matrix** that contains counts of errors. In general, a confusion matrix lists the number of times one thing was confused with another. Thus for example a substitution matrix will be a square matrix of size 26×26 (or more generally $|A| \times |A|$, for an alphabet A) that represents the number of times one letter was incorrectly used instead of another. Following Kernighan et al. (1990) we'll use four confusion matrices.

confusion matrix

del$[x,y]$: count(`xy` typed as `x`)
ins$[x,y]$: count(`x` typed as `xy`)
sub$[x,y]$: count(`x` typed as `y`)
trans$[x,y]$: count(`xy` typed as `yx`)

Note that we've conditioned the insertion and deletion probabilities on the previous character; we could instead have chosen to condition on the following character.

Where do we get these confusion matrices? One way is to extract them from lists of misspellings like the following:

additional: addional, additonal
environments: enviornments, enviorments, enviroments
preceded: preceeded
...

There are lists available on Wikipedia and from Roger Mitton (`http://www.dcs.bbk.ac.uk/~ROGER/corpora.html`) and Peter Norvig (`http://norvig.com/ngrams/`). Norvig also gives the counts for each single-character edit that can be used to directly create the error model probabilities.

An alternative approach used by Kernighan et al. (1990) is to compute the matrices by iteratively using this very spelling error correction algorithm itself. The iterative algorithm first initializes the matrices with equal values; thus, any character is equally likely to be deleted, equally likely to be substituted for any other character, etc. Next, the spelling error correction algorithm is run on a set of spelling errors. Given the set of typos paired with their predicted corrections, the confusion matrices can now be recomputed, the spelling algorithm run again, and so on. This iterative algorithm is an instance of the important **EM** algorithm (Dempster et al., 1977), which we discuss in Chapter 9.

Once we have the confusion matrices, we can estimate $P(x|w)$ as follows (where w_i is the ith character of the correct word w) and x_i is the ith character of the typo x:

$$P(x|w) = \begin{cases} \dfrac{\text{del}[x_{i-1}, w_i]}{\text{count}[x_{i-1}w_i]}, & \text{if deletion} \\ \dfrac{\text{ins}[x_{i-1}, w_i]}{\text{count}[w_{i-1}]}, & \text{if insertion} \\ \dfrac{\text{sub}[x_i, w_i]}{\text{count}[w_i]}, & \text{if substitution} \\ \dfrac{\text{trans}[w_i, w_{i+1}]}{\text{count}[w_i w_{i+1}]}, & \text{if transposition} \end{cases} \tag{5.6}$$

Using the counts from Kernighan et al. (1990) results in the error model probabilities for `acress` shown in Fig. 5.4.

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

Figure 5.5 shows the final probabilities for each of the potential corrections; the unigram prior is multiplied by the likelihood (computed with Eq. 5.6 and the confusion matrices). The final column shows the product, multiplied by 10^9 just for readability.

Candidate Correction	Correct Letter	Error 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 5.5 Computation of the ranking for each candidate correction, using the language model shown earlier and the error model from Fig. 5.4. The final score is multiplied by 10^9 for readability.

The computations in Fig. 5.5 show that our implementation of the noisy channel model chooses `across` as the best correction, and `actress` as the second most likely word.

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.

For this reason, it is important to use larger language models than unigrams. For example, if we use 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:

$$\begin{aligned} \text{P(actress}|\text{versatile)} &= .000021 \\ \text{P(across}|\text{versatile)} &= .000021 \\ \text{P(whose}|\text{actress)} &= .0010 \\ \text{P(whose}|\text{across)} &= .000006 \end{aligned}$$

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

$$\begin{aligned} \text{P(“versatile actress whose”)} &= .000021 * .0010 = 210 \times 10^{-10} \\ \text{P(“versatile across whose”)} &= .000021 * .000006 = 1 \times 10^{-10} \end{aligned}$$

Combining the language model with the error model in Fig. 5.5, the bigram noisy channel model now chooses the correct word *actress*.

Evaluating spell correction algorithms is generally done by holding out a training, development and test set from lists of errors like those on the Norvig and Mitton sites mentioned above.

5.2 Real-word spelling errors

real-word error detection

The noisy channel approach can also be applied to detect and correct **real-word spelling errors**, errors that result in an actual word of English. This can happen from typographical errors (insertion, deletion, transposition) that accidentally produce a real word (e.g., *there* for *three*) or because the writer substituted the wrong spelling of a homophone or near-homophone (e.g., *dessert* for *desert*, or *piece* for *peace*). A number of studies suggest that between 25% and 40% of spelling errors are valid English words as in the following examples (Kukich, 1992):

This used to belong to *thew* queen. They are leaving in about fifteen *minuets* to go to her house.
The design *an* construction of the system will take more than a year.
Can they *lave* him my messages?
The study was conducted mainly *be* John Black.

The noisy channel can deal with real-word errors as well. Let's begin with a version of the noisy channel model first proposed by Mays et al. (1991) to deal with these real-word spelling errors. Their algorithm takes the input sentence $X = \{x_1, x_2, \ldots, x_k, \ldots, x_n\}$, generates a large set of candidate correction sentences $C(X)$, then picks the sentence with the highest language model probability.

To generate the candidate correction sentences, we start by generating a set of candidate words for each input word x_i. The candidates, $C(x_i)$, include every English word with a small edit distance from x_i. With edit distance 1, a common choice (Mays et al., 1991), the candidate set for the real word error *thew* (a rare word meaning 'muscular strength') might be C(thew) = {the, thaw, threw, them, thwe}. We then make the simplifying assumption that every sentence has only one error. Thus the set of candidate sentences $C(X)$ for a sentence X = `Only two of thew apples` would be:

```
only two of thew apples
oily two of thew apples
only too of thew apples
only to of thew apples
only tao of the apples
only two on thew apples
only two off thew apples
only two of the apples
only two of threw apples
only two of thew applies
only two of thew dapples
...
```

Each sentence is scored by the noisy channel:

$$\hat{W} = \underset{W \in C(X)}{\operatorname{argmax}} P(X|W)\,P(W) \tag{5.7}$$

For $P(W)$, we can use the trigram probability of the sentence.

What about the channel model? Since these are real words, we need to consider the possibility that the input word is not an error. Let's say that the channel probability of writing a word correctly, $P(w|w)$, is α; we can make different assumptions about exactly what the value of α is in different tasks; perhaps α is .95, assuming people write 1 word wrong out of 20, for some tasks, or maybe .99 for others. Mays et al. (1991) proposed a simple model: given a typed word x, let the channel model $P(x|w)$ be α when $x = w$, and then just distribute $1-\alpha$ evenly over all other candidate corrections $C(x)$:

$$p(x|w) = \begin{cases} \alpha & \text{if } x = w \\ \dfrac{1-\alpha}{|C(x)|} & \text{if } x \in C(x) \\ 0 & \text{otherwise} \end{cases} \tag{5.8}$$

Now we can replace the equal distribution of $1-\alpha$ over all corrections in Eq. 5.8; we'll make the distribution proportional to the edit probability from the more sophisticated channel model from Eq. 5.6 that used the confusion matrices.

Let's see an example of this integrated noisy channel model applied to a real word. Suppose we see the string `two of thew`. The author might have intended to type the real word `thew` ('muscular strength'). But `thew` here could also be a typo for `the` or some other word. For the purposes of this example let's consider edit distance 1, and only the following five candidates `the`, `thaw`, `threw`, and `thwe` (a rare name) and the string as typed, `thew`. We took the edit probabilities from Norvig's (2009) analysis of this example. For the language model probabilities, we used a Stupid Backoff model (Section **??**) trained on the Google N-grams:

$$\begin{aligned}
\text{P(the}|\text{two of)} &= 0.476012 \\
\text{P(thew}|\text{two of)} &= 9.95051 \times 10^{-8} \\
\text{P(thaw}|\text{two of)} &= 2.09267 \times 10^{-7} \\
\text{P(threw}|\text{two of)} &= 8.9064 \times 10^{-7} \\
\text{P(them}|\text{two of)} &= 0.00144488 \\
\text{P(thwe}|\text{two of)} &= 5.18681 \times 10^{-9}
\end{aligned}$$

Here we've just computed probabilities for the single phrase *two of thew*, but the model applies to entire sentences; so if the example in context was *two of thew*

people, we'd need to also multiply in probabilities for P(people|of the), P(people|of thew), P(people|of threw), and so on.

Following Norvig (2009), we assume that the probability of a word being a typo in this task is .05, meaning that $\alpha = P(w|w)$ is .95. Fig. 5.6 shows the computation.

x	w	x|w	P(x|w)	P(w|w_{i-2}, w_{i-1})	10^8P(x|w)P(w|w_{i-2}, w_{i-1})
thew	the	ew|e	0.000007	0.48	333
thew	thew		α=0.95	9.95×10^{-8}	9.45
thew	thaw	e|a	0.001	2.1×10^{-7}	0.0209
thew	threw	h|hr	0.000008	8.9×10^{-7}	0.000713
thew	thwe	ew|we	0.000003	5.2×10^{-9}	0.00000156

Figure 5.6 The noisy channel model on 5 possible candidates for thew, with a Stupid Backoff trigram language model computed from the Google N-gram corpus and the error model from Norvig (2009).

For the error phrase *two of thew*, the model correctly picks *the* as the correction. But note that a lower error rate might change things; in a task where the probability of an error is low enough (α is very high), the model might instead decide that the word *thew* was what the writer intended.

5.3 Noisy Channel Model: The State of the Art

State of the art implementations of noisy channel spelling correction make a number of extensions to the simple models we presented above.

First, rather than make the assumption that the input sentence has only a single error, modern systems go through the input one word at a time, using the noisy channel to make a decision for that word. But if we just run the basic noisy channel system described above on each word, it is prone to **overcorrecting**, replacing correct but rare words (for example names) with more frequent words (Whitelaw et al. 2009, Wilcox-O'Hearn 2014). Modern algorithms therefore need to augment the noisy channel with methods for detecting whether or not a real word should actually be corrected. For example state of the art systems like Google's (Whitelaw et al., 2009) use a blacklist, forbidding certain tokens (like numbers, punctuation, and single letter words) from being changed. Such systems are also more cautious in deciding whether to trust a candidate correction. Instead of just choosing a candidate correction if it has a higher probability $P(w|x)$ than the word itself, these more careful systems choose to suggest a correction w over keeping the non-correction x only if the difference in probabilities is sufficiently great. The best correction w is chosen only if:

$$\log P(w|x) - \log P(x|x) > \theta$$

autocorrect

Depending on the specific application, spell-checkers may decide to **autocorrect** (automatically change a spelling to a hypothesized correction) or merely to flag the error and offer suggestions. This decision is often made by another classifier which decides whether the best candidate is good enough, using features such as the difference in log probabilities between the candidates (we'll introduce algorithms for classification in the next chapter).

Modern systems also use much larger dictionaries than early systems. Ahmad and Kondrak (2005) found that a 100,000 word UNIX dictionary only contained

73% of the word types in their corpus of web queries, missing words like *pics, multiplayer, google, xbox, clipart*, and *mallorca*. For this reason modern systems often use much larger dictionaries automatically derived from very large lists of unigrams like the Google N-gram corpus. Whitelaw et al. (2009), for example, used the most frequently occurring ten million word types in a large sample of web pages. Because this list will include lots of misspellings, their system requires a more sophisticated error model. The fact that words are generally more frequent than their misspellings can be used in candidate suggestion, by building a set of words and spelling variations that have similar contexts, sorting by frequency, treating the most frequent variant as the source, and learning an error model from the difference, whether from web text (Whitelaw et al., 2009) or from query logs (Cucerzan and Brill, 2004). Words can also be automatically added to the dictionary when a user rejects a correction, and systems running on phones can automatically add words from the user's address book or calendar.

We can also improve the performance of the noisy channel model by changing how the prior and the likelihood are combined. In the standard model they are just multiplied together. But often these probabilities are not commensurate; the language model or the channel model might have very different ranges. Alternatively for some task or dataset we might have reason to trust one of the two models more. Therefore we use a weighted combination, by raising one of the factors to a power λ:

$$\hat{w} = \operatorname*{argmax}_{w \in V} P(x|w)\, P(w)^{\lambda} \tag{5.9}$$

or in log space:

$$\hat{w} = \operatorname*{argmax}_{w \in V} \log P(x|w) + \lambda \log P(w) \tag{5.10}$$

We then tune the parameter λ on a development test set.

confusion sets

Finally, if our goal is to do real-word spelling correction only for specific **confusion sets** like *peace/piece*, *affect/effect*, *weather/whether*, or even grammar correction examples like *among/between*, we can train supervised classifiers to draw on many features of the context and make a choice between the two candidates. Such classifiers can achieve very high accuracy for these specific sets, especially when drawing on large-scale features from web statistics (Golding and Roth 1999, Lapata and Keller 2004, Bergsma et al. 2009, Bergsma et al. 2010).

5.3.1 Improved Edit Models: Partitions and Pronunciation

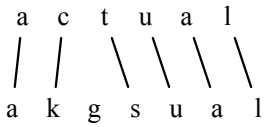
Other recent research has focused on improving the channel model $P(t|c)$. One important extension is the ability to compute probabilities for multiple-letter transformations. For example Brill and Moore (2000) propose a channel model that (informally) models an error as being generated by a typist first choosing a word, then choosing a partition of the letters of that word, and then typing each partition, possibly erroneously. For example, imagine a person chooses the word `physical`, then chooses the partition `ph y s i c al` She would then generate each partition, possible with errors. For example the probability that she would generate the string `fisikle` with partition `f i s i k le` would be $p(\texttt{f}|\texttt{ph}) * p(\texttt{i}|\texttt{y}) * p(\texttt{s}|\texttt{s}) * p(\texttt{i}|\texttt{i}) * p(\texttt{k}|\texttt{k}) * p(\texttt{le}|\texttt{al})$. Unlike the Damerau-Levenshtein edit distance, the Brill-Moore channel model can thus model edit probabilities like $P(\texttt{f}|\texttt{ph})$ or $P(\texttt{le}|\texttt{al})$, or the high likelihood of $P(\texttt{ent}|\texttt{ant})$. Furthermore, each edit is conditioned on where

it is in the word (*beginning, middle, end*) so instead of $P(\texttt{f}|\texttt{ph})$ the model actually estimates $P(\texttt{f}|\texttt{ph}, \text{beginning})$.

More formally, let R be a partition of the typo string x into adjacent (possibly empty) substrings, and T be a partition of the candidate string. Brill and Moore (2000) then approximates the total likelihood $P(x|w)$ (e.g., P(`fisikle`|`physical`)) by the probability of the single best partition:

$$P(x|w) \approx \max_{R,T\, s.t. |T|=|R|} \sum_{i=1}^{|R|} P(T_i|R_i, \text{position}) \tag{5.11}$$

The probability of each transform $P(T_i|R_i)$ can be learned from a training set of triples of an error, the correct string, and the number of times it occurs. For example given a training pair `akgsual`/`actual`, standard minimum edit distance is used to produce an alignment:



This alignment corresponds to the sequence of edit operations:

`a`→`a`, `c`→`k`, ε→`g` `t`→`s`, `u`→`u`, `a`→`a`, `l`→`l`

Each nonmatch substitution is then expanded to incorporate up to N additional edits; For N=2, we would expand `c`→`k` to:

`ac`→`ak`
`c`→`cg`
`ac`→`akg`
`ct`→`kgs`

Each of these multiple edits then gets a fractional count, and the probability for each edit $\alpha \rightarrow \beta$ is then estimated from counts in the training corpus of triples as $\frac{\text{count}(\alpha\rightarrow\beta)}{\text{count}(\alpha)}$.

Another research direction in channel models is the use of **pronunciation** in addition to spelling. Pronunciation is an important feature in some non-noisy-channel algorithms for spell correction like the GNU **aspell** algorithm (Atkinson, 2011), which makes use of the metaphone pronunciation of a word (Philips, 1990). Metaphone is a series of rules that map a word to a normalized representation of its pronunciation. Some example rules:

aspell

- "Drop duplicate adjacent letters, except for C."
- "If the word begins with 'KN', 'GN', 'PN', 'AE', 'WR', drop the first letter."
- "Drop 'B' if after 'M' and if it is at the end of the word"

Aspell works similarly to the channel component of the noisy channel model, finding all words in the dictionary whose pronunciation string is a short edit distance (1 or 2 pronunciation letters) from the typo, and then scoring this list of candidates by a metric that combines two edit distances: the pronunciation edit distance and the weighted letter edit distance.

Pronunciation can also be incorporated directly the noisy channel model. For example the Toutanova and Moore (2002) model, like aspell, interpolates two channel models, one based on spelling and one based on pronunciation. The pronunciation

function SOUNDEX(*name*) **returns** *soundex form*

1. Keep the first letter of *name*
2. Drop all occurrences of non-initial a, e, h, i, o, u, w, y.
3. Replace the remaining letters with the following numbers:
   b, f, p, v → 1
   c, g, j, k, q, s, x, z → 2
   d, t → 3
   l → 4
   m, n → 5
   r → 6
4. Replace any sequences of identical numbers, only if they derive from two or more letters that were *adjacent* in the original name, with a single number (e.g., 666 → 6).
5. Convert to the form `Letter Digit Digit Digit` by dropping digits past the third (if necessary) or padding with trailing zeros (if necessary).

Figure 5.7 The Soundex Algorithm

letter-to-sound
phones

model is based on using **letter-to-sound** models to translate each input word and each dictionary word into a sequences of **phones** representing the pronunciation of the word. For example `actress` and `aktress` would both map to the phone string `ae k t r ix s`. See Chapter 32 on the task of letter-to-sound or **grapheme-to-phoneme**.

deduplication

Some additional string distance functions have been proposed for dealing specifically with **names**. These are mainly used for the task of **deduplication** (deciding if two names in a census list or other namelist are the same) rather than spell-checking.

The Soundex algorithm (Knuth 1973, Odell and Russell 1922) is an older method used originally for census records for representing people's names. It has the advantage that versions of the names that are slightly misspelled will still have the same representation as correctly spelled names. (e.g., Jurafsky, Jarofsky, Jarovsky, and Jarovski all map to J612). The algorithm is shown in Fig. 5.7.

Jaro-Winkler

Instead of Soundex, more recent work uses **Jaro-Winkler** distance, which is an edit distance algorithm designed for names that allows characters to be moved longer distances in longer names, and also gives a higher similarity to strings that have identical initial characters (Winkler, 2006).

Bibliographical and Historical Notes

Algorithms for spelling error detection and correction have existed since at least Blair (1960). Most early algorithms were based on similarity keys like the Soundex algorithm (Odell and Russell 1922, Knuth 1973). Damerau (1964) gave a dictionary-based algorithm for error detection; most error-detection algorithms since then have been based on dictionaries. Early research (Peterson, 1986) had suggested that spelling dictionaries might need to be kept small because large dictionaries contain very rare words (wont, veery) that resemble misspellings of other words, but Damerau and Mays (1989) found that in practice larger dictionaries proved more helpful. Damerau (1964) also gave a correction algorithm that worked for single errors.

The idea of modeling language transmission as a Markov source passed through a noisy channel model was developed very early on by Claude Shannon (1948).

The idea of combining a prior and a likelihood to deal with the noisy channel was developed at IBM Research by Raviv (1967), for the similar task of **optical character recognition** (**OCR**). While earlier spell-checkers like Kashyap and Oommen (1983) had used likelihood-based models of edit distance, the idea of combining a prior and a likelihood seems not to have been applied to the spelling correction task until researchers at AT&T Bell Laboratories (Kernighan et al. 1990, Church and Gale 1991) and IBM Watson Research (Mays et al., 1991) roughly simultaneously proposed noisy channel spelling correction. Much later, the Mays et al. (1991) algorithm was reimplemented and tested on standard datasets by Wilcox-O'Hearn et al. (2008), who showed its high performance.

Most algorithms since Wagner and Fischer (1974) have relied on dynamic programming.

Recent focus has been on using the web both for language models and for training the error model, and on incorporating additional features in spelling, like the pronunciation models described earlier, or other information like parses or semantic relatedness (Jones and Martin 1997, Hirst and Budanitsky 2005).

See Mitton (1987) for a survey of human spelling errors, and Kukich (1992) for an early survey of spelling error detection and correction. Norvig (2007) gives a nice explanation and a Python implementation of the noisy channel model, with more details and an efficient algorithm presented in Norvig (2009).

Exercises

5.1 Suppose we want to apply add-one smoothing to the likelihood term (channel model) $P(x|w)$ of a noisy channel model of spelling. For simplicity, pretend that the only possible operation is deletion. The MLE estimate for deletion is given in Eq. 5.6, which is $P(x|w) = \frac{\text{del}[x_i 1, w_i]}{\text{count}(x_{i1} w_i)}$. What is the estimate for $P(x|w)$ if we use add-one smoothing on the deletion edit model? Assume the only characters we use are lower case `a-z`, that there are V word types in our corpus, and N total characters, not counting spaces.

Ahmad, F. and Kondrak, G. (2005). Learning a spelling error model from search query logs. In *HLT-EMNLP-05*, pp. 955–962.

Atkinson, K. (2011). Gnu aspell..

Bergsma, S., Lin, D., and Goebel, R. (2009). Web-scale n-gram models for lexical disambiguation.. In *IJCAI-09*, pp. 1507–1512.

Bergsma, S., Pitler, E., and Lin, D. (2010). Creating robust supervised classifiers via web-scale n-gram data. In *ACL 2010*, pp. 865–874.

Blair, C. R. (1960). A program for correcting spelling errors. *Information and Control*, *3*, 60–67.

Brill, E. and Moore, R. C. (2000). An improved error model for noisy channel spelling correction. In *ACL-00*, Hong Kong, pp. 286–293.

Church, K. W. and Gale, W. A. (1991). Probability scoring for spelling correction. *Statistics and Computing*, *1*(2), 93–103.

Cucerzan, S. and Brill, E. (2004). Spelling correction as an iterative process that exploits the collective knowledge of web users. In *EMNLP 2004*, Vol. 4, pp. 293–300.

Damerau, F. J. (1964). A technique for computer detection and correction of spelling errors. *Communications of the ACM*, *7*(3), 171–176.

Damerau, F. J. and Mays, E. (1989). An examination of undetected typing errors. *Information Processing and Management*, *25*(6), 659–664.

Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum likelihood from incomplete data via the *EM* algorithm. *Journal of the Royal Statistical Society*, *39*(1), 1–21.

Golding, A. R. and Roth, D. (1999). A Winnow based approach to context-sensitive spelling correction. *Machine Learning*, *34*(1-3), 107–130.

Hirst, G. and Budanitsky, A. (2005). Correcting real-word spelling errors by restoring lexical cohesion. *Natural Language Engineering*, *11*, 87–111.

Jones, M. P. and Martin, J. H. (1997). Contextual spelling correction using latent semantic analysis. In *ANLP 1997*, Washington, D.C., pp. 166–173.

Kashyap, R. L. and Oommen, B. J. (1983). Spelling correction using probabilistic methods. *Pattern Recognition Letters*, *2*, 147–154.

Kernighan, M. D., Church, K. W., and Gale, W. A. (1990). A spelling correction program base on a noisy channel model. In *COLING-90*, Helsinki, Vol. II, pp. 205–211.

Knuth, D. E. (1973). *Sorting and Searching: The Art of Computer Programming Volume 3*. Addison-Wesley.

Kukich, K. (1992). Techniques for automatically correcting words in text. *ACM Computing Surveys*, *24*(4), 377–439.

Lapata, M. and Keller, F. (2004). The web as a baseline: Evaluating the performance of unsupervised web-based models for a range of NLP tasks. In *HLT-NAACL-04*.

Mays, E., Damerau, F. J., and Mercer, R. L. (1991). Context based spelling correction. *Information Processing and Management*, *27*(5), 517–522.

Mitton, R. (1987). Spelling checkers, spelling correctors and the misspellings of poor spellers. *Information processing & management*, *23*(5), 495–505.

Norvig, P. (2007). How to write a spelling corrector. http://www.norvig.com/spell-correct.html.

Norvig, P. (2009). Natural language corpus data. In Segaran, T. and Hammerbacher, J. (Eds.), *Beautiful data: the stories behind elegant data solutions*. O'Reilly.

Odell, M. K. and Russell, R. C. (1918/1922). U.S. Patents 1261167 (1918), 1435663 (1922). Cited in Knuth (1973).

Peterson, J. L. (1986). A note on undetected typing errors. *Communications of the ACM*, *29*(7), 633–637.

Philips, L. (1990). Hanging on the metaphone. *Computer Language*, *7*(12).

Raviv, J. (1967). Decision making in Markov chains applied to the problem of pattern recognition. *IEEE Transactions on Information Theory*, *13*(4), 536–551.

Shannon, C. E. (1948). A mathematical theory of communication. *Bell System Technical Journal*, *27*(3), 379–423. Continued in the following volume.

Toutanova, K. and Moore, R. C. (2002). Pronunciation modeling for improved spelling correction. In *ACL-02*, Philadelphia, PA, pp. 144–151.

Veblen, T. (1899). *Theory of the Leisure Class*. Macmillan Company, New York.

Wagner, R. A. and Fischer, M. J. (1974). The string-to-string correction problem. *Journal of the Association for Computing Machinery*, *21*, 168–173.

Whitelaw, C., Hutchinson, B., Chung, G. Y., and Ellis, G. (2009). Using the web for language independent spellchecking and autocorrection. In *EMNLP-09*, pp. 890–899.

Wilcox-O'Hearn, L. A. (2014). Detection is the central problem in real-word spelling correction. http://arxiv.org/abs/1408.3153.

Wilcox-O'Hearn, L. A., Hirst, G., and Budanitsky, A. (2008). Real-word spelling correction with trigrams: A reconsideration of the Mays, Damerau, and Mercer model. In *CICLing-2008*, pp. 605–616.

Winkler, W. E. (2006). Overview of record linkage and current research directions. Tech. rep., Statistical Research Division, U.S. Census Bureau.