An Improved Error Model for Noisy Channel Spelling Correction

Eric Brill and Robert C. Moore
Microsoft Research
One Microsoft Way
Redmond, WA 98052
{brill, bobmoore}@microsoft.com

Abstract

The noisy channel model has been applied to a wide range of problems, including spelling correction. These models consist of two components: the source model and the channel model. Very little research has gone into improving the channel model for spelling correction. This paper describes a new channel model for spelling correction, based on generic string-to-string edits. Using this model gives significant performance improvements compared to previously proposed models.

Introduction

The noisy channel model (Shannon 1948) has been successfully applied to a wide range of problems, including spelling correction. These models consist of two components: the source model and the channel model. People have devoted considerable energy to improving both components, with resulting improvements in overall system accuracy. However, relatively little research has gone into improving the channel model for spelling correction. This paper describes an improvement to noisy channel spelling correction: a more powerful model of spelling errors, beyond typing mistakes or cognitive errors that have previously been employed. Our model works by learning generic string-to-string edits along with the probabilities of each of these edits. This more powerful model gives significant improvements in accuracy over previous approaches to noisy channel spelling correction.

1 Noisy Channel Spelling Correction

This paper will address the problem of automatically training a system to correct generalizing word spelling errors. We do not address the problem of correcting specific word set confusions such as {to, too, two} (see (Golding and Roth 1999)). We will define the spelling correction problem abstractly as follows: Given an alphabet \( \Sigma \), a dictionary \( D \) consisting of strings in \( \Sigma^* \) and a string \( s \), where \( s \not\in D \) and \( s \in \Sigma^* \), find the word \( w \in D \) that is most likely to have been erroneously input as \( s \). The requirement that \( s \not\in D \) can be dropped, but it only makes sense to do so in the context of a sufficiently powerful language model.

In a probabilistic system, we want to find \( \text{argmax}_w P(w \mid s) \) Applying Bayes' Rule and dropping the constant denominator, we get the unnormalized posterior: \( \text{argmax}_w P(s \mid w) P(w) \). Now we have a noisy channel model for spelling correction with two components: the source model \( P(w) \) and the channel model \( P(s \mid w) \).

The model assumes that natural language text is generated as follows: First, a person chooses a word according to the probability distribution \( P(w) \). Then the person attempts to output the word, but the noisy channel induces some errors. The errors are modeled by string-to-string edits, which are learned along with the probabilities of each of these edits.
outputstrings instead, according to the distribution \(P(s | w)\). For instance, under typical circumstances we would expect \(P(\text{the} | \text{the})\) to be relatively high, \(P(\text{hippopotamus} | \text{the})\) to be extremely low.

In this paper, we will refer to the channel model as the error model.

Two seminal papers first posed a noisy channel model solution to the spelling correction problem in (Mayes, Damerau et al. 1991), using bigrams to represent the source model. The first defines the confusion set of a string by including all words \(w\) in the dictionary \(D\) such that \(s\) can be derived from \(w\) by a single application of one of the four edit operations:

1. Adding a single letter.
2. Deleting a single letter.
3. Replacing one letter with another.
4. Transposing two adjacent letters.

Let \(C\) be the number of words in the confusion set of \(d\). Then they define the error model for all \(s\) in the confusion set of \(d\) as:

\[
P(s | d) = \begin{cases} 
\alpha & \text{if } s = d \\
(1 - \alpha) \frac{1}{C-1} & \text{otherwise}
\end{cases}
\]

This is a very simple error model, where the probability of being correct is uniform across all other words in the confusion set.

Church and Gale (1991) propose a more sophisticated error model. Like Mayes, Damerau, et al. (1991), they consider candidate source words only those words that are minimal with respect to the set of words defined above. However, two improvements are made.

First, instead of assigning equal probability to each unique edit operation, the probability of inserting or deleting character is conditioned on the letter appearing immediately to the left of that character.

The error probabilities are derived by first assuming all edits are equiprobable. Then, a training corpus of spaced-delimited strings is found in a large collection of text that do not appear in the dictionary. For each word, more than one data word does not appear in the dictionary. They iteratively run the spellchecker over the training corpus and use the corrections to update the error probabilities. Ristad and Yianilos (1997) present another algorithm for deriving these probabilities from a training corpus, and show that for the problem of word pronunciation using single character edits, the learned string distance gives one fourth the error rate compared to using unweighted edits.

2. An Improved Error Model

Previous error models have been based on Damerau-Levenshtein distance measures (Damerau 1964; Levenshtein 1966), where the distance between two strings is the minimum number of single character insertions, substitutions, and deletions (and in some cases character pair transpositions) necessary to transform one string into another. Improvements have been made by associating probabilities with individual edit operations.

We propose a much more generic error model. Let \(\Sigma\) be an alphabet. Our model allows edit operation of the form \(\alpha \to \beta\) where \(\alpha, \beta \in \Sigma\). The probability that when a user intends to type the string \(\alpha\) instead of \(\beta\), the probability of inserting \(\beta\) instead is conditioned on the edit operation allowed. Church and Gale (1991), Mayes Damerau et al. (1991), and Ristad and Yianilos (1997) properly
In addition, we condition on the position of the string to be altered, $P(\alpha \rightarrow \beta | PSN)$, where $PSN$ is the source dictionary word. The position determines the location of substring $\alpha$ in the source dictionary word. Positional information is a powerful conditioning feature for rich edit operations. For instance, $P(e | a)$ does not vary greatly between the three positions mentioned above. However, $P(ant | ent)$ is highly dependent upon position. People rarely mistype *antler* as *entler*, but often mistype *reluctant* as *reluctent*. Within the noisy channel framework, we can informally think of our error model as follows. First, a person chooses a word to generate. Then, she picks a partition of the characters of that word. Then, she types each partition, possibly erroneously. For example, a person might choose to generate the word *physical*. She would then pick a partition from the set of all possible partitions: say: $\text{phys}$ $\text{i}$ $\text{ical}$. Then, she would generate each partition possibly erroneous. After choosing the partition and type the word *physical*, the probability of generating the string *fisikle* with the partition *f*$ $\text{i}$ $\text{s}$ $\text{i}$ $\text{k}$ $\text{le}$ is given by $P(f | \text{ph}) P(i | y) P(s | s) P(i | i) P(k | c) P(le | al)$. The above example points to advantages of our model compared to previous models based on weighted Damerau-Levenshtein distance. Not that neither $P(\text{ph})$ nor $P(|\text{h}|)$ were modeled directly. A number of studies have pointed out that a high percentage of misspelled words are wrong due to single letter insertions/insertion/deletions or from letter pair transposition (Damerau 1964; Peterson 1986). However, even if this idea does not imply that nothing is gained by modeling more powerful edit operations, if somebody types the string *confidant* we do not really want to model how the error occurs. For example, the antecedent can more accurately be modeled $P(\text{ant} | \text{ente})$. By taking a more generic approach, modeling can more accurately model the errors people make.

A formal presentation of our model follows. Let $\text{Part}(w)$ be the set of all possible ways of partitioning string $w$ into adjacent (possibly null) substrings. For a particular partition $R \in \text{Part}(w)$, where $|R| = j$, let $R_i$ be the $i$th segment of $R$. Then, $P(s | w)$ can be written as:

$$P(s | w) = \sum_{R \in \text{Part}(w)} \prod_{R_i} P(R_i | w)$$

One particular alignment of $s$ and $w$ induces a set of edits that derive $s$ from $w$. By only considering the best partitioning, we simplify this to:

$$P(s | w) = \max_{R \in \text{Part}(w)} \prod_{T \in \text{Part}(s)} P(R | w) P(T | R_i)$$

We do not have a good way to derive $P(R)$ and, in running experiments we determined that poorly modeling it gave slightly worse performance than not modeling it at all. We dropped this term.

3 Training Model

To train the model, we need a training set consisting of $\{s_i, w_i\}$ pairs, representing spelling errors paired with the correct spelling of the word. We
aligning letters with those in based on minimizing the distance between and based on single character insertions, deletions and substitutions for instance given the training pair: "aksual", actual, actual could be aligned as:

\[
\begin{align*}
\text{a} & \rightarrow \text{a} \\
\text{u} & \rightarrow \text{k} \\
\text{t} & \rightarrow \text{g} \\
\text{u} & \rightarrow \text{s} \\
\text{a} & \rightarrow \text{u} \\
\text{l} & \rightarrow \text{a}
\end{align*}
\]

This corresponds to the sequence of edit operations:

\[
\begin{align*}
\text{a} & \rightarrow \text{a} \\
\text{k} & \rightarrow \text{g} \\
\text{u} & \rightarrow \text{s} \\
\text{a} & \rightarrow \text{u} \\
\text{l} & \rightarrow \text{a}
\end{align*}
\]

To allow for richer contextual information, we expand each nonmatch substitution to incorporate up to \(N\) additional adjacent edits. For example, for the first nonmatch edit in the example above, with \(N=2\), we would generate the following substitutions:

\[
\begin{align*}
\text{c} & \rightarrow \text{k} \\
\text{a} & \rightarrow \text{g} \\
\text{c} & \rightarrow \text{g} \\
\text{c} & \rightarrow \text{g} \\
\text{c} & \rightarrow \text{g} \\
\text{c} & \rightarrow \text{g} \\
\text{c} & \rightarrow \text{g} \\
\text{c} & \rightarrow \text{g} \\
\text{c} & \rightarrow \text{g} \\
\end{align*}
\]

We would do similarly for the other nonmatch edits, and give each of these substitutions fractional count. We then calculate the probability of each substitution \(\alpha \rightarrow \beta\) as \(\text{count}(\alpha \rightarrow \beta) / \text{count}(\alpha \rightarrow \beta)\) as explained above. We have derived from our training data the spelling errors found in the corpus, and then used those errors for training. Counting \(\alpha \rightarrow \beta\) would simply be the number of times \(\alpha\) occurs in the text of the training corpus and give an associated corpus weight of 1.

(a) From a large collection of representative text, count the number of occurrences of \(\alpha\).

(b) Adjust our estimate of the rate at which people make typing errors.

Since the errors vary widely and are difficult to measure, we can only crudely approximate it. Fortunately, we have found empirically that the rate is not very sensitive to the value chosen.

Essentially, we are doing one iteration of the Expectation-Maximization algorithm (Dempster, Laird et al. 1977). The context that is useful will accumulate a fractional count across multiple instances where the context that is noise will accumulate a significant count.

4 Applying the Model

Given a string where \(s \notin D\) we want to return \(\arg\max_w P(w \mid s) P(w \mid \text{context})\). Our approach will be to return the best of candidate according the error model and then score these candidate taking into account the source probabilities.

We are given a dictionary and a set of parameters where each parameter is \(P(\alpha \rightarrow \beta)\) for some \(\alpha, \beta \in \Sigma^*\), meaning the probability that if a string intended the noisy channel will produce \(\beta\) instead of \(\alpha\). First note that for a particular pair of strings \(s, w\) we can use the standard dynamic programming algorithm for finding the edit distance by filling a \(|s| \times |w|\) weight matrix (Wagner and Fisher 1974; Hall and Dowling 1980), without minor changes. For computing the Damerau-Levenshtein distance between two strings, this can be done in \(O(|s| \times |w|)\) time. When allowing generic edit operations, the complexity increases to \(O(|s| \times 2^{|w|})\) filling cells (i,j) in the matrix for computing the Damerau-Levenshtein distance. We only examine cells \((i,j-1), (i-1,j)\) and \((i-1,j-1)\). With generic edits, we have examined cells \((a,b)\) where \(a \leq i \leq b \leq j\).
We first precompile the dictionary into a trie, with each node in the trie corresponding to a vector of weights. Each node in the trie corresponds to a column in the weight matrix associated with computing the distance between and the string prefix ending at that trie node.

We store the $\alpha$ parameters as a trie of tries. We have one trie corresponding to all strings $\alpha$ that appear on the left hand side of some substitution in our parameter set. At every node in this trie, corresponding to a string $\alpha$, we point to a trie consisting of all strings $\beta$ that appear on the right hand side of substitution parameters with $\alpha$ on the left hand side. We store the substitution probabilities at the terminal nodes the tries.

By storing both $\alpha$ and $\beta$ strings in reverse order, we can efficiently compute edit distance over the entire dictionary. We process the dictionary trie from the root downwards, filling in the weight vector at each node. To find the substitution parameters that are applicable, given a particular node and a particular position in the input string $s$ (this corresponds to filling in one cell in the vector of weights), we trace from the node to the root, while tracing down the trie from the root. As we trace down the trie, we encounter terminal nodes. For each terminal node, we follow the pointer to the corresponding $\beta$ trie, and then trace backwards from the position in $s$ while tracing down the $\beta$ trie.

Note that searching through a static dictionary is not a requirement of our error model. It is possible that with different search techniques we could apply our model to languages such as Turkish for which static dictionaries are inappropriate (Oflazer 1994).

Given a 200,000-word dictionary and using the best error model, we are able to spell check strings not in the dictionary in approximately 50 milliseconds on average, running on a Dell 1500mhz Pentium III workstation.

5 Results

5.1 Error Model

We experimented using a 0,000-word corpus of common English spelling errors, paired with their correct spellings. We used 80% of this corpus for training and 20% for evaluation. Our dictionary contained approximately 200,000 entries, including all words in the test set. The results in this section are obtained with a language model that assigns uniform probability to all words in the dictionary.

In Table 1 we show $K$-best results for different maximum context window sizes, without using positional information. For instance, the 2-best accuracy is the percentage of time the correct answer is one of the top two answers returned by the system. Note that a maximum window of zero corresponds to the single character insertion, deletion, and substitution edit distances, weighted with their probabilities. We see that beyond a point, additional context provides us with more accurate spellings, and beyond that, additional context neither helps nor hurts.

| Max Window | 1-Best | 2-Best | 3-Best |
|------------|--------|--------|--------|
| 0          | 87.0   | 93.9   | 95.9   |
| CG         | 89.5   | 94.9   | 96.5   |
| 1          | 90.9   | 95.6   | 96.8   |
| 2          | 92.9   | 97.1   | 98.1   |
| 3          | 93.6   | 97.4   | 98.5   |
| 4          | 93.6   | 97.4   | 98.5   |

Table Results without positional information
In Table 1, the row labeled CG shows the results when we allow the equivalent set of edit operations to those used in (Church and Gale 1991). This properly subset of the CG model, in which the maximum window size is zero and a proper subset of the edit set is used, is essentially equivalent to the Church and Gale error model except that the models do not allow the use of arbitrary numbers of edits. The CG model is essentially equivalent to the Church and Gale error model, except

(a) the models above can posit an arbitrary number of edits and

(b) we did not do parameter reestimation (see below).

Next, we measured how much we gain by conditioning on the position of the edit relative to the source word. These results are shown in Table 2. As we expected, positional information helps more when using a richer edit set than when using only single character edits. For a maximum window size of 0, using positional information gives a 13% relative improvement in 1-best accuracy, whereas for a maximum window size of 5, the gain is 22%. Our full strength model gives 2% relative error reduction in 1-best accuracy compared to the CG model (95.0% compared to 93.9%).

| Max Window | 1-Best | 2-Best | 3-Best |
|------------|--------|--------|--------|
| 0          | 88.7   | 95.1   | 96.6   |
| 1          | 92.8   | 96.5   | 97.4   |
| 2          | 94.6   | 98.0   | 98.7   |
| 3          | 95.0   | 98.0   | 98.8   |
| 4          | 95.0   | 98.0   | 98.8   |
| 5          | 95.1   | 98.0   | 98.8   |

Table 2: Results with positional information.

We experimented with iteratively reestimating parameters on the original formulation (Church and Gale 1991). Doing so resulted in a slight degradation in performance. The data we are using is cleaner than that used (Church and Gale 1991) which probably explains why estimation benefited in their experiments and did not give any benefit to the error models in our experiments.

5.2 Adding a Language Model

Next, we explore what happens to our results as we add a language model to the Brown Corpus and found occurrences of all words used in experiments. We mapped these words to incorrect spellings they were paired with the second best word in our spellchecker's correct spellings. We used two language models. The first assumed all words are equally likely, i.e., the null language model used above. The second used a trigram language model derived from a large collection of online text (not including the Brown Corpus). Because spellcheckers typically applied right after a word was typed, the language model only used left context.

We show the results in Figure 1, where we used the error model with positional information and a maximum context window of four, and used the language model to score the word candidates returned by the error model. Note that the best language model, the results are with the results quoted above, gives a 1-best score of 95.0%, compared to 93.9% in Figure 2. Because results for the Brown Corpus were computed per token, whereas above were computed per type. One question we wanted to ask is whether using a good language model would obviate the need for a good error model. Figure 2 shows that a language model improves results, using the better model in Figure 1 still gives significantly better results. Using a language model with no best error model gives 3.6% error reduction compared to using a language model with the CG error model. Rescoring the 20-best output of the CG model instead of the 5-best only
improves the best accuracy from 90.9% to 91.0%.

![Figure 1: Spelling Correction Improvement When Using a Language Model](image1)

![Figure 2: Using the CG Error Model with a Trigram Language Model](image2)

**Conclusion**

We have presented a new error model for noisy channel spelling correction based on generic string to string edits, and have demonstrated that it results in significant improvement compared to previous approaches. Without a language model, our error model gives a 52% reduction in spelling correction error rate compared to the weighted Damerau-Levenshtein distance technique of Church and Gale. With a language model, our model gives a 4% reduction in error.

One exciting future line of research is to explore error models that adapt to individual or subpopulation of edits. Without a rich set of edits, we hope highly accurate individualized spell checking can soon become reality.

**References**

Church, K. and W. Gale (1991). "Probability scoring for spell correction." *Statistical Computing* 3:93-103.

Damerau, F. (1964). "A technique for computer detection and correction of spelling errors." *Communications of the ACM* 7(3):659-664.

Dempster, A., N. Laird, et al. (1977). "Maximum likelihood from incomplete data via the EM algorithm." *Journal of the Royal Statistical Society* 39(1):1-21.

Golding, A. and D. Roth (1999). "A Winnow-Based Approach to Spelling Correction." *Machine Learning* 34:87-130.

Hall, P. and G. Dowling (1980). "Approximate string matching." *ACM Computing Surveys* 12(4):7-38.

Jurafsky, D. and J. Martin (2000). *Speech and Language Processing*. Prentice Hall.

Kukich, K. (1992). "Techniques for automatically correcting words in text." *ACM Computing Surveys* 24(4):437-439.

Levenshtein, V. (1966). "Binary codes capable of correcting deletions, insertions and reversals." *Soviet Physics – Doklady* 10:707-710.
Mayes, E., Damerau, F., et al. (1991). "Context-Based Spelling Correction." Information Processing and Management 27(5): 517-522.

Oflazer, K. (1994). Spelling Correction in Agglutinative Languages. Applied Natural Language Processing Stuttgart Germany.

Peterson, J. (1986). "A note on undetected typing errors." Communications of the ACM 29(7): 633-637.

Ristad, E. and P. Yianilos (1997). Learning String Edit Distance. International Conference on Machine Learning Morgan Kaufmann.

Shannon, C. (1948). "A mathematical theory of communication." Bell System Technical Journal 27(3): 79-423.

Wagner, R. and M. Fisher (1974). "The string to string correction problem." JACM 21: 68-173.