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: a source model and a 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: a source model and a channel model. For many applications, 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 via a more powerful model of spelling errors, be they typing mistakes or cognitive errors, than has 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 generic single word spelling errors.[1] 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 Σ, a dictionary D consisting of strings in Σ^* and a string s, where $s \notin D$ and $s \in \Sigma^*$, find the word $w \in D$ that is most likely to have been erroneously input as s. The requirement that $s \notin D$ 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: $\text{argmax}_w\ P(s \mid w)^*P(w)$. We now have a noisy channel model for spelling correction, with two components, the source model P(w) and the channel model P(s | w). The model assumes that natural language text is generated as follows: First a person chooses a word to output, according to the probability distribution P(w). Then the person attempts to output the word w, but the noisy channel induces the person to output string s instead, according to the

[1] Two very nice overviews of spelling correction can be found in (Kukich 1992) and (Jurafsky and Martin 2000).

distribution P(s | w). For instance, under typical circumstances we would expect P(the | the) to be very high, P(teh | the) to be relatively high and P(hippopotamus | 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), word bigrams are used for the source model. For the error model, they first define the *confusion set* of a string s to include s, along with 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) Add a single letter.
(2) Delete a single letter.
(3) Replace one letter with another.
(4) Transpose 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:

$$P(s \mid d) = \begin{cases} \alpha \text{ if s = d} \\ \dfrac{(1\text{-}\alpha)}{(C-1)} \text{otherwise} \end{cases}$$

This is a very simple error model, where α is the prior on a typed word being correct, and the remaining probability mass is distributed evenly among 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 as candidate source words only those words that are a single basic edit away from s, using the same edit set as above. However, two improvements are made. First, instead of weighing all edits equally, each unique edit has a probability associated with it. Second, insertion and deletion probabilities are conditioned on context. The probability of inserting or deleting a 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. They use as a training corpus a set of space-delimited strings that were found in a large collection of text, and that (a) do not appear in their dictionary and (b) are no more than one edit away from a word that does appear in the dictionary. They iteratively run the spell checker over the training corpus to find corrections, then use these corrections to update the edit probabilities. Ristad and Yianilos (1997) present another algorithm for deriving these edit probabilities from a training corpus, and show that for the problem of word pronunciation, using the learned string edit distance gives one fourth the error rate compared to using unweighted edits.

2 An Improved Error Model

Previous error models have all 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 derive one string from another. Improvements have been made by associating probabilities with individual edit operations.

We propose a much more generic error model. Let Σ be an alphabet. Our 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 the string α they type β instead. Note that the edit operations allowed in Church and Gale (1991), Mayes, Damerau et al. (1991) and Ristad and Yianilos (1997), are properly subsumed by our generic string to string substitutions.

In addition, we condition on the position in the string that the edit occurs in, $P(\alpha \rightarrow \beta \mid PSN)$, where PSN = {start of word, middle of word, end of word}.[2] The position is determined by the location of substring α 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(ent | ant) 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 picks 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: **ph y s i c al**. Then she would generate each partition, possibly with errors. After choosing this particular word and partition, the probability of generating the string ***fisikle*** with the partition **f i s i k le** would be P(f | ph) *P(i | y) * P(s | s) *P(i | i) * P(k | c) *P(le | al).[3]

The above example points to advantages of our model compared to previous models based on weighted Damerau-Levenshtein distance. Note that neither P(f | ph) nor P(le | al) are modeled directly in the previous approaches to error modeling. A number of studies have pointed out that a high percentage of misspelled words are wrong due to a single letter insertion, substitution, or deletion, or from a letter pair transposition (Damerau 1964; Peterson 1986). However, even if this is the case, it does not imply that nothing is to be gained by modeling more powerful edit operations. If somebody types the string *confidant*, we do not really want to model this error as P(a | e), but rather P(ant | ent). And *anticedent* can more accurately be modeled by P(anti | ante), rather than P(i | e). By taking a more generic approach to error modeling, we can more accurately model the errors people make.

A formal presentation of our model follows. Let 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 $|R|=j$ (R consists of j contiguous segments), let R_i be the i^{th} segment. Under our model,

$$P(s \mid w) = \sum_{R \in Part(w)} P(R \mid w) \sum_{\substack{T \in Part(s) \\ |T|=|R|}} \prod_{i=1}^{|R|} P(T_i \mid R_i)$$

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

$$P(s \mid w) = \max_{R \in \text{Part}(w), T \in \text{Part}(s)} P(R|w) \prod_{i=1}^{|R|} P(T_i|R_i)$$

We do not yet have a good way to derive P(R | w), and in running experiments we determined that poorly modeling this distribution gave slightly worse performance than not modeling it at all, so in practice we drop this term.

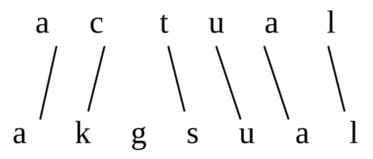
3 Training the Model

To train the model, we need a training set consisting of $\{s_i, w_i\}$ string pairs, representing spelling errors s_i paired with the correct spelling of the word w_i. We begin by aligning the letters in s_i with those in w_i based on minimizing the edit distance

[2] Another good PSN feature would be *morpheme boundary*.

[3] We will leave off the positional conditioning information for simplicity.

between s_i and w_i, based on single character insertions, deletions and substitutions. For instance, given the training pair <akgsual, actual>, this could be aligned as:



This corresponds to the sequence of edit operations:

a→a c→k ε→g t→s u→u a→a l→l

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:

c → k
ac → ak
c → kg
ac → akg
ct → kgs

We would do similarly for the other nonmatch edits, and give each of these substitutions a fractional count.

We can then calculate the probability of each substitution α → β as count(α → β)/count(α). count(α → β) is simply the sum of the counts derived from our training data as explained above. Estimating count(α) is a bit tricky. If we took a text corpus, then extracted all the spelling errors found in the corpus and then used those errors for training, count(α) would simply be the number of times substring α occurs in the text corpus. But if we are training from a set of {s_i, w_i} tuples and not given an associated corpus, we can do the following:

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

(b) Adjust the count based on an estimate of the rate with which people make typing errors.

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

Essentially, we are doing one iteration of the Expectation-Maximization algorithm (Dempster, Laird et al. 1977). The idea is that contexts that are useful will accumulate fractional counts across multiple instances, whereas contexts that are noise will not accumulate significant counts.

4 Applying the Model

Given a string s, where $s \notin D$, we want to return $\mathrm{argmax}_w\, P(w \mid s) P(w \mid context)$. Our approach will be to return an n-best list of candidates according to the error model, and then rescore these candidates by taking into account the source probabilities.

We are given a dictionary D and a set of parameters P, where each parameter is P(α → β) for some $\alpha, \beta \in \Sigma^*$, meaning the probability that if a string α is intended, the noisy channel will produce β instead. First note that for a particular pair of strings {s, w} we can use the standard dynamic programming algorithm for finding edit distance by filling a |s|*|w| weight matrix (Wagner and Fisher 1974; Hall and Dowling 1980), with only minor changes. For computing the Damerau-Levenshtein distance between two strings, this can be done in O(|s|*|w|) time. When we allow generic edit operations, the complexity increases to $O(|s|^2 * |w|^2)$. In filling in a cell (i,j) in the matrix for computing Damerau-Levenshtein distance we need only examine cells (i,j-1), (i-1,j) and (i-1,j-1). With generic edits, we have to examine all cells (a,b) where $a \le i$ and $b \le j$.

We first precompile the dictionary into a trie, with each node in the trie corresponding to a vector of weights. If we think of the x-axis of the standard weight matrix for computing edit distance as corresponding to w (a word in the dictionary), then the vector at each node in the trie corresponds to a column in the weight matrix associated with computing the distance between s and the string prefix ending at that trie node.

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

By storing both α and β 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 in the trie and a particular position in the input string s (this corresponds to filling in one cell in one vector of a dictionary trie node) we trace up from the node to the root, while tracing down the α trie from the root. As we trace down the α trie, if we encounter a terminal node, we follow the pointer to the corresponding β trie, and then trace backwards from the position in s while tracing down the β trie.

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

Given a 200,000-word dictionary, and using our best error model, we are able to spell check strings not in the dictionary in approximately 50 milliseconds on average, running on a Dell 610 500mhz Pentium III workstation.

5 Results

5.1 Error Model in Isolation

We ran experiments using a 10,000-word corpus of common English spelling errors, paired with their correct spelling. 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 set of single character insertion, deletion and substitution edits, weighted with their probabilities. We see that, up to a point, additional context provides us with more accurate spelling correction 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 1 Results without positional information

In Table 1, the row labelled CG shows the results when we allow the equivalent set of edit operations to those used in (Church and Gale 1991). This is a proper superset of the set of edits where the maximum window is zero and a proper subset of the edits where the maximum window is one. 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 4, the gain is 22%. Our full strength model gives a 52% relative error reduction on 1-best accuracy compared to the CG model (95.0% compared to 89.5%).

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, as was done in the original formulation in (Church and Gale 1991). Doing so resulted in a slight degradation in performance. The data we are using is much cleaner than that used in (Church and Gale 1991) which probably explains why reestimation benefited them 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. In order to get errors in context, we took the Brown Corpus and found all occurrences of all words in our test set. Then we mapped these words to the incorrect spellings they were paired with in the test set, and ran our spell checker to correct the misspellings. 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 on-line text (not including the Brown Corpus). Because a spell checker is typically applied right after a word is 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 with a maximum context window of four, and used the language model to rescore the 5 best word candidates returned by the error model. Note that for the case of no language model, the results are lower than the results quoted above (e.g. a 1-best score above of 95.0%, compared to 93.9% in the graph). This is because the results on the Brown Corpus are computed per *token*, whereas above we were computing results per *type.*

One question we wanted to ask is whether using a good language model would obviate the need for a good error model. In Figure 2, we applied the trigram model to resort the 5-best results of the CG model. We see that while a language model improves results, using the better error model (Figure 1) still gives significantly better results. Using a language model with our best error model gives a 73.6% 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 1-best accuracy from 90.9% to 91.0%.

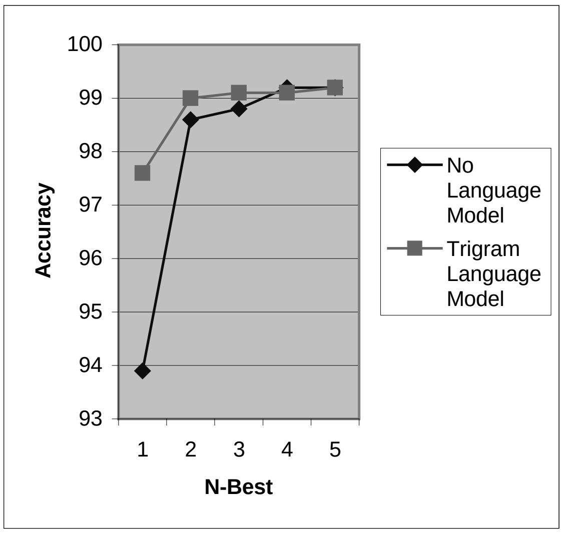


Figure 1 Spelling Correction Improvement When Using a Language Model

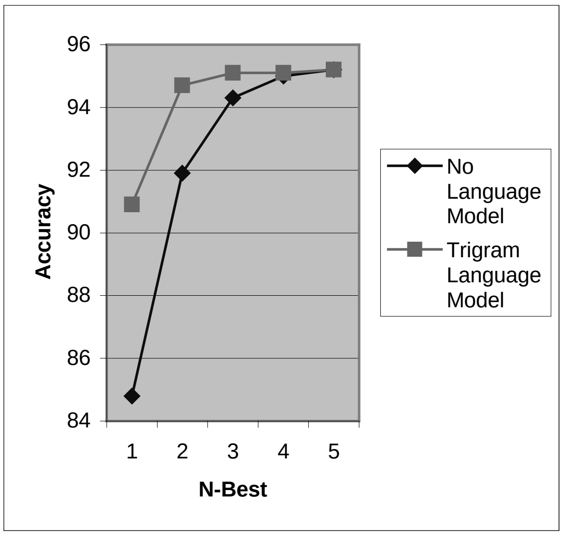


Figure 2 Using the CG Error Model with a Trigram Language Model

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 a significant improvement in performance 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 74% reduction in error.

One exciting future line of research is to explore error models that adapt to an individual or subpopulation. With a rich set of edits, we hope highly accurate individualized spell checking can soon become a reality.

References

Church, K. and W. Gale (1991). "Probability Scoring for Spelling Correction." Statistics and Computing **1**: 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**: 107-130.

Hall, P. and G. Dowling (1980). "Approximate string matching." ACM Computing Surveys **12**(4): 17-38.

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): 377-439.

Levenshtein, V. (1966). "Binary codes capable of correcting deletions, insertions and reversals." Soviet Physice -- Doklady **10**: 707-710.

Mayes, E., F. Damerau, 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): 379-423.

Wagner, R. and M. Fisher (1974). “The string to string correction problem.” JACM **21**: 168-173.