Statistics and Computing (1991) **1**, 93–103

Probability scoring for spelling correction

KENNETH W. CHURCH and WILLIAM A. GALE

AT&T Bell Laboratories, 600 Mountain Ave., Murray Hill, NJ, USA

Received August 1990 and accepted December 1990

This paper describes a new program, CORRECT, which takes words rejected by the Unix® SPELL program, proposes a list of candidate corrections, and sorts them by probability score. The probability scores are the novel contribution of this work. They are based on a noisy channel model. It is assumed that the typist knows what words he or she wants to type but some noise is added on the way to the keyboard (in the form of typos and spelling errors). Using a classic Bayesian argument of the kind that is popular in recognition applications, especially speech recognition (Jelinek, 1985), one can often recover the intended correction, c, from a typo, t, by finding the correction c that maximizes $Pr(c)\ Pr(t \mid c)$. The first factor, $Pr(c)$, is a prior model of word probabilities; the second factor, $Pr(t \mid c)$, is a model of the noisy channel that accounts for spelling transformations on letter sequences (insertions, deletions, substitutions and reversals). Both sets of probabilities were estimated using data collected from the Associated Press (AP) newswire over 1988 and 1989 as a training set. The AP generates about 1 million words and 500 typos per week.

In evaluating the program, we found that human judges were extremely reluctant to cast a vote given only the information available to the program, and that they were much more comfortable when they could see a concordance line or two. The second half of this paper discusses some very simple methods of modeling the context using n-gram statistics. Although n-gram methods are much too simple (compared with much more sophisticated methods used in artificial intelligence and natural language processing), we have found that even these very simple methods illustrate some very interesting estimation problems that will almost certainly come up when we consider more sophisticated models of contexts. The problem is how to estimate the probability of a context that we have not seen. We compare several estimation techniques and find that some are useless. Fortunately, we have found that the Good–Turing method provides an estimate of contextual probabilities that produces a significant improvement in program performance. Context is helpful in this application, but only if it is estimated very carefully.

At this point, we have a number of different knowledge sources—the prior, the channel and the context—and there will certainly be more in the future. In general, performance will be improved as more and more knowledge sources are added to the system, as long as each additional knowledge source provides some new (independent) information. As we shall see, it is important to think more carefully about combination rules, especially when there are a large number of different knowledge sources.

Keywords: Automated learning, spelling correction, n-gram language model, Good–Turing estimates

1. The problem

The CORRECT program reads a list of misspelled words from the input stream (stdin) and prints a set of candidate corrections for each word on the output stream (stdout). CORRECT also produces a probability estimate along with each candidate correction (unless there is only one candidate correction). These probabilities distinguish CORRECT from previous spelling correctors. The problem of finding just the right way to estimate these probabilities from the various available sources of knowledge poses a technical challenge for both statistics and artificial intelligence. In this paper we have only dealt with a few of the more obvious possibilities.

Table 1. *Examples of output of* CORRECT

Typo	Corrections (%)
detered	deterred (100) metered (0) petered (0)
laywer	lawyer (100) layer (0) lawer (0)
negotations	negotiations
notcampaigning	???*
progession	progression (94) procession (4) profession (2)
usuallly	usually
winky	windy (69) wink (20) winks (7) kinky (2) wonky (1) pinky (1) dinky (0) winy (0) inky (0)

*??? indicates that no correction was found.

Table 1 presents some sample output produced by the Unix® command, 'spell paper | correct', where *paper* is a text file containing the misspelled words in column 1.

2. Proposing candidate corrections

The first stage of CORRECT searches a wordlist to find candidate corrections that differ from the input typo§ by a single insertion, deletion, substitution or reversal. The wordlist was collected from five sources, the AP newswire, the Unix® SPELL program, the *Cobuild* dictionary (Sinclair *et al*, 1987), the *Collins Dictionary of the English Language* (Hanks *et al*, 1979), and *Roget's International Thesaurus* (Chapman, 1977). This first stage produces the output shown in Table 2, given the input typo, 'acress'.

The candidate 'actress', for example, can be mistyped as 'acress' by deleting the 't' at position 2. The symbols @ and # represent nulls. (The transformations are named from the point of view of the correct word, not the typo.) The typo, 'acress', can arise from 'acres' by insertion of 's' after either the fourth or the fifth letter, so 'acres' appears twice in the table as a candidate correction. This unusually difficult example was selected to illustrate the four transformations; most examples do not have so many high-scoring possibilities.

Table 2. *Example of candidate corrections*

Typo	Correction	Transformation			
acress	actress	@	t	2	deletion
acress	cress	a	#	0	insertion
acress	caress	ac	ca	0	reversal
acress	access	r	c	2	substitution
acress	across	e	o	3	substitution
acress	acres	s	#	4	insertion
acress	acres	s	#	5	insertion

§For the purposes of this experiment, a typo is a lower-case word rejected by the Unix® SPELL program.

Table 3. *An example from the deletion table*

Key	Correction	
grea	t	4
gret	a	3
grat	e	2
geat	r	1
reat	g	0

In principle, one could implement the transformations very straightforwardly by trying all the possibilities. The insertion operation is actually implemented this way. It requires n dictionary accesses to see if the nth letter in the typo may have been inserted. Unfortunately, the deletion operation is much more expensive; there are 26 letters that might have been deleted in $n+1$ positions. In order to reduce the number of dictionary accesses, the system makes use of a precomputed table as shown in Table 3. This table maps words missing a letter to corrections. With this table, the system can check for deletions in one table look-up. The table is also useful for checking for substitutions and reversals. Of course, there is a cost in space. The deletion table has about a million entries, approximately ten times as many as the dictionary. The deletion table is stored using heuristic hashing methods very similar to those in SPELL (McIlroy, 1982).

3. Scoring

Two versions of CORRECT have been studied: one with context and the other without context. We discuss the simpler no-context version first.

Each candidate correction is scored by the Bayesian combination rule $Pr(c)\,Pr(t\,|\,c)$, and then normalized by the sum of the scores for all proposed candidates. Care must be taken in estimating the prior because of sparse data problems. It is possible (and even likely) that a proposed correction might not have appeared in the training set. Some methods of estimating the prior would produce undesirable results in this case. For example, the maximum likelihood estimate (MLE) would estimate $Pr(c) = 0$ and, consequently, many candidate corrections would be rejected just because they did not happen to appear in the training set (the 1988 AP corpus). We will encounter even more severe forms of the sparse data problem when we consider context.

We will consider three estimation methods that deal with the sparse data problems in three different ways. All