Fifty years of spellchecking

Journal Article

Fifty years of spellchecking

Roger Mitton
Department of Computer Science and Information Systems,
Birkbeck, University of London

By the standards of the computing industry, spellchecking has a long history. It began in the late fifties - the days of mainframes and punched paper tape – an early publication is (Blair, 1960), and an oft-cited paper is (Damerau, 1964).

Most of the methods used a dictionary (meaning, in this context, simply a list of correct spellings) but some did not. One system (Morris and Cherry, 1975), when presented with a text for checking, split it up into three-letter sequences (trigrams), counted the number of each, and then calculated an "index of peculiarity" for each word, based on the frequency of the trigrams it contained, finally drawing the user's attention to the more peculiar-looking ones. The typo *exmination*, for example, contains *exm* and *xmi*, trigrams probably not shared by any other word in the text, so it would be rated as rather peculiar and would appear near the top of the list. Of course the user still had the job of spotting the errors in this list, and many misspellings do not contain unusual trigrams and so would not figure in the list at all, but it often succeeded in highlighting a typo. And, being dictionary-free, it would work just as well for, say, Spanish or Greek.

But most systems checked a text by looking up all the words in a dictionary. Publishers were beginning to make use of computer technology, and dictionaries for spellchecking could be extracted from the machine-readable versions of the published ones. A big problem, even into the eighties, was the small size of computer memories. Holding an entire dictionary in main memory (the rapid-access part of the computer's storage) was out of the question. The dictionary had to be held on disc and small portions of it read into main memory as required. Consequently much ingenuity went into compressing the dictionary.

One technique used was affix-stripping (McIlroy, 1982). Instead of storing *computes, computed, computing, computer, computers, computable, computability, computation, computational*, you store just *compute*, and have a set of rules that strip suffixes and adjust the stem if necessary. Having derived *compute* from, say, *computability*, and having found *compute* in the dictionary, you conclude that *computability* is an acceptable word. You can do the same with prefixes, deriving *civil* from *uncivil*. There needs to be some ordering of the rules, to accept *undoubtedly* but not *undoubtlyed* and some way of handling words that look as if they have affixes but don't, such as *prosper, seabed* and *farthing*. Though effective for the checking part of a spellchecker's task, this was less useful for suggesting the correct spelling, since simply adding affixes to a stem runs the risk of generating non-existent words – *doubtedly? undoubting?*

Another consequence of holding the dictionary on disc was to make the checking a slow process (by computer standards) since a disc access is thousands of times slower than a main-memory access. A partial solution was to hold, in main memory, a list of the most frequent words in the language. If a spellchecker was checking the first

sentence of this paragraph, and it held just the most frequent one hundred words of written English in main memory, it would find *of, the, on, was, to, a, by, is* and *than* (i.e. two-fifths of the tokens) without having to consult the main dictionary; if it held the next few hundred, it would find *another, make, since* and *main* (Leech et al., 2001).

There was some debate about whether a spellchecker's dictionary should be large or small. "The larger, the better," might be one's first reaction. But it was pointed out that mistyping a short word can often produce another word (Peterson, 1986), and that people sometimes write one word for another – bigger *then* me, the *principle* function, the teacher *tort* us (Mitton, 1987). These real-word errors are, of course, not detected by dictionary look-up, and the spellchecker is more likely to let them through if its dictionary is full of rare words. So perhaps the dictionary should not be too big?

A study of this problem established, however, that, when people use a rare word, it is very likely to be a correct spelling and not a real-word error, so that a spellchecker with a small dictionary, while it might occasionally detect a real-word error, would more often be raising false alarms over correctly spelt, rare words (Damerau and Mays, 1989). While this is clearly true for a large proportion of rare words – it's unlikely that someone who writes *okapi* did not mean *okapi* – there is a subset of rare words that bear a strong resemblance to common words, and the occurrence of one of these words is in fact more likely to be an error than a correct use. *Calender* (with an *e*), for example, is in the dictionary (it's a machine for smoothing cloth or paper), and it occurs fourteen times in the BNC (British National Corpus), but all fourteen are misspellings of *calendar* (though one of the two occurrences of *calenders* (plural) is correct) (Mitton et al. 2007). Similarly, *withe (with), ail (all), tor (for), canvasses (canvases), posses (possess), polices (policies), abut (about), wold (world/would/wild)* and *rime (time)*. So, while a larger dictionary is generally preferable, rare words that resemble common words should be treated as potential errors.

Correction, as opposed to the detection, of errors consisted of generating a list of words that somewhat resembled the error. An early algorithm, described in (Peterson, 1980), aimed to reverse any of the possible processes that might have given rise to a single-letter typo. Take, for example, the misspelling *pord*. The typist might have inserted an extra letter, so let's look up *ord, prd, pod* and *por*. Or the typist might have transposed two adjacent letters, so look up *oprd, prod* and *podr*. Perhaps one letter was substituted for another, so look up *aord, bord, cord* and so on, then *pard, pbrd, pcrd* ..., then *poad, pobd*..., and *pora, porb* ... down to *porz*. And finally, let's do something similar on the assumption that the typist omitted a letter: *apord, bpord, cpord, ..., paord, pbord, ..., poard, pobrd, ..., porad, porbd, ...*, ending with *porda* to *pordz*. If any of these variations turn out to be in the dictionary, you offer them to the user: *pod, prod, cord, ford, lord, word, pond, pore, pork, porn, port, pored*. (There are more efficient ways of achieving the same result – see, for example Oflazer (1996), Savary (2002) or Mihov and Schulz (2004).)

This list is in no particular order – there is no notion of the spellchecker's best guess, second-best and so on. And it is restricted, obviously, to single-letter errors. This is not too serious for mistyped words, the great majority of which contain just one single-letter error (Pollock and Zamora, 1984), but it is less useful for misspellings; it would not offer the right word, for example, for "Mother pord the tea."

Before the arrival of the PC in the early eighties, wordprocessors were mostly used by secretaries, who were sent on training courses to learn how to use them, so the spellcheckers of the time were designed for people whose spelling was assumed to be pretty good. Generating a long list of suggestions, possibly containing some very obscure words, was seen as more important than ordering the list in a helpful way. The earlier PC-based spellcheckers continued this policy. When offered *cort*, for example, in, say, "I've cort a cold," WordPerfect 5.1 (circa 1985) responded with the following: *cart, cert, coat, colt, cont, coot, copt, cor, cord, corf, cork, corm, corn, corp, corr, cors, corti, cost, cot, court, crt, curt, carat, carate, card, cared, caret, carried, carrot, carte, cerate, cered, ceroid, chaired, charade, chard, chariot, charred, chart, cheered, cheroot, chert, chirred, chord, choreoid, chorioid, choroid, cirrate, cored, corrade, corrode, corrupt, coward, cowered, curate, curd, cured, curet, curette, curried, karate, kart, keyword, scared, scarred, scirrhoid, scored, scoured, scurried.*

The complaints that most people had with spellcheckers, however, were not with the lists of suggestions but with shortcomings in error detection. On the one hand, the spellchecker would flag names, newly coined words and technical terms as errors – this could be ameliorated by allowing users to build up private dictionaries to be used as supplements to the spellchecker's main dictionary. On the other hand, the spellchecker failed to flag real-word errors, and this was a serious defect since errors of this kind are surprisingly common – several studies, admittedly of handwritten text, found that a quarter to a third of all misspelt words were real-word errors (Wing and Baddeley, 1980; Sterling, 1983; Mitton, 1987; Brooks et al. 1993). Hence a little poem that was in circulation, in different versions:

> I have a spelling chequer; it came with my pea sea.
> It plainly marques for my revue miss takes eye cannot sea.
> I've run this poem threw it, I'm sure yore pleased two no.
> It's letter perfect in its weigh – my chequer tolled me sew.

An approach to this problem, developed in the eighties and nineties, was to use confusion sets (e.g. Golding, 1995; Golding and Roth, 1999). A confusion set is a small set of words – usually two but sometimes three or four – that are likely to be confused with one another, such as *{there, their, they're}* or *{principle, principal}*. You provide the spellchecker with a list of confusion sets. It then scans the text, looking for any of the words in the list. Let's say the text contains the sentence, "The sand-eel is the principle food for many birds and animals." Having found an occurrence of *principle*, it assesses whether any of the other members of the confusion set (here just *principal*) would be more appropriate in that position. It might make this assessment on the basis of syntax, semantics, collocation or any other information it might have. If it decided that *principal* would be more appropriate here, it would flag *principle* as an error and propose *principal* as a correction.

It is important to find a way to calibrate these assessments of appropriateness and only to flag an error if the spellchecker is confident of its assessment, since the great majority of occurrences of the words in confusion sets are in fact correct, and a spellchecker that was constantly raising false alarms would be irritating and effectively unusable.

Unfortunately, the early research with this technique used a small list of about twenty confusion sets, and most of the subsequent experiments used the same set, to preserve comparability with earlier work, though (Carlson et al., 2001) scaled it up to 265. Though sufficient for proof of concept, this small list would obviously be inadequate for use in a real-life spellchecker. Only recently, however, has a serious attempt been made to produce a list sufficiently large to tackle unrestricted text (Pedler and Mitton, 2010). Experiments with a test corpus of errors collected from student essays, online bulletin boards and so on, suggest that, with a list of about 6,000 confusion sets, a spellchecker could detect around 70% of the real-word errors.

The rapid take-up of PCs in the eighties meant that the use of computers, and particularly of wordprocessors, was no longer confined to professionals. Users could no longer be assumed to be good spellers; in fact they increasingly looked to the spellchecker to help them with their spelling. Poor spellers do not want a list of fifty suggestions, with the required word buried (or possibly not) somewhere in the middle; they want a short list of about half a dozen with the required word preferably at the top.

To produce such a list, a spellchecker can begin by assembling a large set of possible candidates, perhaps some hundreds of them. These are words that somewhat resemble the error – perhaps they begin with the same letter, share a couple of consonants in common and are roughly the same length. Each of these candidates is then compared with the error, using some string-matching algorithm. Many algorithms have been proposed, but a simple one would be to count the number of letters or letter-pairs that the candidate has in common with the misspelling. This provides a kind of measurement of how close each of the candidates is to the misspelling, and the spellchecker offers the best few to the user.

A simple system like this works quite well for a large proportion of misspellings, matching *bicycle* to, say, *bycicyle*. But it works less well for the misspellings of poor spellers; for *cort*, it would favour *court*, *cert* or *corm*, though *caught* might well be the target.

Another string-matching algorithm is based on the notion of edit-distance (Levenshtein, 1966; Wagner and Fischer, 1974). In its simplest form, you take the misspelling on the one hand and one of the candidates on the other, and you work out how many single-letter changes are required to change the one into the other, where a single-letter change could be the insertion of a letter or the omission of a letter or the changing of one letter into another. (In many systems, the transposition of two adjacent letters is also counted as a single-letter change.) For example, if the misspelling was *yot* and the candidate was *yoke*, you could get from *yoke* to *yot* by changing the *k* to a *t* and omitting the *e* – two changes, so the edit-distance is 2.

The lower the edit-distance, the closer the match. You calculate the edit-distance for each of the candidates and then present them in order, lowest first. So if, say, we had three candidates for *yot – yoke, pot* and *yacht* – we would calculate their edit-distance to *yot* to be 2 for *yoke*, 1 for *pot* and 3 for *yacht*, so we would present them in the order *pot, yoke, yacht*.

In a more elaborate version of edit-distance (Veronis, 1988; Mitton, 1996), you attach costs to each of the single-letter changes; a low cost would be attached to a relatively trivial change, such as doubling a consonant (e.g. *harrass* for *harass*), but a high one to an unlikely change, such as changing a *p* to a *y*. Let's suppose we attached the following costs in our *yot* example:

pot	*p* to *y* unlikely, say cost of 5	Total: 5
yoke	*k* to *t* unlikely, say cost of 5; addition of e not uncommon, say 2	Total: 7
yacht	*a* to *o* not surprising given the pronunciation, say 1; likewise the omission of the *ch*, say 2	Total: 3

So we would present these candidates in the order *yacht, pot, yoke*.

The costs can be held in a general table applicable to all words – you might decide, for instance, that changing a *p* to a *y* will always have a cost of 5, while changing a *c* to a *k* will cost 3. Or they can take account of the immediate context, e.g. changing a *p* to an *f* is normally improbable, say cost of 4 or 5, but if it's the *p* in *ph*, it's a lot more likely, say cost of 2. Or they can be tailored for individual words – omitting the *t* from *mortal* would attract a high cost, but omitting it from *mortgage* a much lower one.

This system enables a spellchecker to anticipate the sort of misspellings that are caused by the quirks of English orthography; it can make allowance for the *ch* of *yacht*, the *c* of *scissors* or the *w* of *answer*. Although these examples arise from the mismatch of spelling and pronunciation, as many misspellings do, the system can deal with other sorts of misspelling. *Rember*, for example, is a common misspelling of *remember*, so we attach a low cost to the omission of the *em*. *Latest* is sometimes written *lastest*, so we attach a low cost to the insertion of an *s*. (For more detail see Mitton (2008).)

By means of these and other techniques, spellcheckers became quite good at offering the required word at the head of the list, and this, paradoxically, gave rise to a new sort of misspelling – the Cupertino. These are caused by people, whether from an excess of faith or a lack of attention, choosing the first suggestion from the spellchecker's list without looking very closely, thus producing sentences such as, "The Wine Bar Company is opening a chain of brassieres," or, "The nightwatchman threw the switch and eliminated the backyard." They are called Cupertinos because a version of Microsoft Word did not have the spelling *cooperation* in its dictionary, only the hyphenated *co-operation*. If someone typed *cooperation*, it would, bizarrely, offer *Cupertino*, the name of a suburban city in California, as its first suggestion. There are documents on the web containing phrases such as "agreement on bilateral Cupertino".

When I began my research into spellchecking in the 1980's, I gave a presentation on my ideas to my colleagues in the Department of Computer Science at Birkbeck, and they asked why I did not adopt the simple and direct approach of assembling a very large database of misspellings and mapping each one onto its target word. When you found a misspelling in the text you were checking, you would just look it up in this database and find the target word that it was matched with. I replied that no such

collection of misspellings existed, that it would be an enormous job to create one and, given the inventiveness that people bring to the creation of misspellings, it would be an unmanageably huge database. Twenty five years on and something very like this database now exists, thanks to the internet and the big search engines.

The search engine companies – Google, Yahoo and the rest – keep a log of all the queries that people key in, and, since they have been doing this for several years and since millions of people use these engines, the log files are enormous. Many of the queries, of course, contain misspellings. There is, therefore, the possibility of implementing my colleagues' suggestion, or something like it.

The spellchecking task that faces a search engine is not the same as that faced by a regular spellchecker. Rather than checking a text of at least a few sentences, the search engine is trying to correct a query consisting of just a few words. The range of possible target words is much wider than for a regular spellchecker, including names of people, places, companies and products. Consequently the dictionary, central to traditional spellchecking, is less useful for query checking; someone who types in *Limp Biscuit* is probably not interested in biscuits but is trying to find out about the rock group *Limp Bizkit*.

One technique that has been described (Cucerzan and Brill, 2004) makes use of the observation that misspellings follow a certain pattern. Around each correct spelling there is an extended family of potential misspellings, some of them bearing a close resemblance to the target, others more remote. The closer the family resemblance, the more common the misspelling. In other words, near-misses are quite common, whereas weird misspellings, though there may be a lot of them altogether, are individually quite rare. If, for example, you asked a hundred secondary-school children to spell the word *scissors*, the most frequent effort would be the correct spelling, then you would find quite a lot of *sissors* and a few each of *siccors, scisors, siscors, sisers* and *sissers*, and then lots of wilder variations, such as *cezzous, saciarres, sisions* and *sorriors*, but only one or two of each (Mitton, 1996).

Given a misspelled query (i.e. it does not correspond to any of the search engine's index terms) – let's call it Q1 – the query checker looks for a match, or a near match, in the log of past queries. This may itself be a misspelled query – call it Q2 – in which case the checker repeats the process, looking for a near-match to Q2 which has also appeared more frequently in the log and is therefore likely to be a closer approximation to the desired search term. This may need to be repeated two or three times until the next nearest match is not another misspelling but a valid search term, as in the following example:

Q1: anol scwartegger
Q2: arnold schwartnegger
Q3: arnold schwarznegger
Q4: arnold schwarzenegger (the required search term)

Whether this technique can be transferred to your own computer depends on the future of computing. You certainly could not accommodate, on your laptop, the gigantic files required to hold the logs, even supposing that the search engine companies were prepared to part with them. But it may be that the personal computer

of the future will do very little processing in its own right but rather will act as your connection into the huge computing power of the internet, so that the spellchecking of your document, along with many other processes, will not actually take place inside your own machine but will be carried out elsewhere, with your machine just showing you the results.

So perhaps, when you make a spelling error and the correct spelling pops into your computer from who knows where ("cloud computing" is the term currently given to this sort of internet based computing), it may be that you will be benefitting not so much from the efforts of good spellers who have gone before you, patiently creating dictionaries, but from the efforts of bad ones, misspelling the same word in a thousand different ways.

References

Blair, C.R. (1960). A program for correcting spelling errors. *Information and Control*, **3**: 60-7.

Brooks, G., Gorman, T. and Kendall, L. (1993). *Spelling it out: the spelling abilities of 11- and 15-year-olds*. Slough: National Foundation for Educational Research.

Carlson, A.J., Rosen, J., and Roth, D. (2001). Scaling up context-sensitive text correction. In *Proceedings of the 13th Innovative Applications of Artificial Intelligence Conference*, Menlo Park, CA.: AAAI Press, 45-50.

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

Damerau, F.J. (1964). A technique for computer detection and correction of spelling errors. *Communications of the A.C.M.*, **7**: 171-6.

Damerau, F.J. and Mays, E. (1989). An examination of undetected typing errors. *Information Processing and Management*, **25** (6): 659-64.

Golding, A.R. (1995). A Bayesian hybrid method for context-sensitive spelling correction. In *Proceedings of the Third Workshop on Very Large Corpora*, Cambridge MA.: Massachusetts Institute of Technology, 39-53.

Golding, A.R. and Roth, D. (1999). A Winnow-based approach to context-sensitive spelling correction. *Machine Learning*, **34**: 107-30.

Leech, G., Rayson, P. and Wilson, A. (2001). *Word Frequencies in Written and Spoken English*. London: Longman.

Levenshtein, V.I. (1966). Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics – Doklady* **10** (8): 707-10.

McIlroy, M.D. (1982). Development of a spelling list. *IEEE Transactions on Communications*, **COM-30** (1): 91-9.

Mihov, S. and Schulz, K.U. (2004). Fast approximate search in large dictionaries. *Computational Linguistics*, **30** (4): 451-77.

Mitton, R. (1987). Spelling checkers, spelling correctors and the misspellings of poor spellers. *Information Processing and Management*, **23** (5): 495-505.

Mitton, R. (1996). *English Spelling and the Computer*. London: Longman.

Mitton, R. (2008). Ordering the suggestions of a spellchecker without using context. *Natural Language Engineering*, **15** (2): 173-92.

Mitton, R., Harrison, D. and Pedler, J. (2007). BNC! Handle with care! Spelling and tagging errors in the BNC. In Davies, M., Rayson, P., Hunston, S. and Danielsson,

P. (eds), *Proceedings of the Corpus Linguistics Conference CL2007*, University of Birmingham, ucrel.lancs.ac.uk/publications/CL2007/.
Morris, R. and Cherry, L.L. (1975). Computer detection of typographical errors. *IEEE Transactions on Professional Communication*, **PC-18** (1): 54-64.
Oflazer, K. (1996). Error tolerant finite-state recognition with applications to morphological analysis and spelling correction. *Computational Linguistics*, **22** (1): 73-89.
Pedler, J. and Mitton, R. (2010). A large list of confusion sets for spellchecking assessed against a corpus of real-word errors. In *Language Research and Evaluation Conference LREC2010*, Malta.
Peterson, J.L. (1980). Computer programs for detecting and correcting spelling errors. *Communications of the A.C.M.*, **23** (12): 676-87.
Peterson, J.L. (1986). A note on undetected typing errors. *Communications of the A.C.M.*, **29** (7): 633-7.
Pollock, J.L. and Zamora, A. (1984). Automatic spelling correction in scientific and scholarly text. *Communications of the A.C.M.*, **27** (4): 358-68.
Savary, A. (2002). Typographical nearset-neighbour search in a finite-state lexicon and its application to spelling correction. In Watson B.W. and Woods, D. (eds), *Proceedings of the 6th International Conference on the Implementation and Application of Automata. Lecture Notes in Computer Science 2494*. Berlin: Springer, 251-60.
Sterling, C.M. (1983). Spelling errors in context. *British Journal of Psychology*, **74**: 353-64.
Veronis, J. (1988). Computerized correction of phonographic errors. *Computers and the Humanities*, **22**: 43-56.
Wagner, R.A., and Fischer, M.J. (1974). The string-to-string correction problem. *Journal of the A.C.M.*, **21** (1): 168-73.
Wing, A.M. and Baddeley, A.D. (1980). Spelling errors in handwriting: a corpus and a distributional analysis. In Frith U. (ed.), *Cognitive Processes in Spelling*. London: Academic Press, 251-85.