SPELLCHECKERS

Roger Mitton

This chapter is an extended version of an earlier paper (Mitton, 2010), by kind permission of the editor.

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 oft-cited paper is (Damerau, 1964). Accounts of various approaches can be found in (Kukich, 1992; Mitton, 1996; Jurafsky and Martin, 2000).

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 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.

(In the earlier work, the term "trigram" usually refers to a sequence of three letters, but, in work of the last two decades, it more often means a sequence of three words; the same applies to "unigram", "bigram" and "n-gram".)

Using a dictionary

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 stored just <compute>, and had a set of rules that stripped suffixes and adjusted the stem if necessary. Having derived <compute> from, say, <computability>, and having found <compute> in the dictionary, you concluded that <computability> was an acceptable word. You could do the same with prefixes, deriving <civil> from <uncivil>.

There needed 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 <whimper>, <seabed> and <farthing> (compare <jumper>, <combed> and <seething>). 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). Spellcheckers relied on dictionary look-up to detect errors. A spellchecker that had <tort> in its dictionary would not spot the error in 'the teacher tort us'. But if we removed <tort> from its dictionary, it would. So perhaps we should remove rare words from the dictionary? Perhaps a spellchecker would work better with a smaller dictionary?

A study of this problem established, however, that, when people use a rare word, it is more likely to be a correct spelling than an error; 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. <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 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.

Suggesting corrections

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 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 try every letter of the alphabet in place of each of the letters of <pord>, from <aord>, <bord>, <cord> and so on, through <pard>, <pbrd> etc. eventually down to <pory> and <porz>. And finally, do something similar on the assumption that the typist omitted a letter, trying every letter inserted at every position, from <apord> down 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) or Mihov and Schulz (2004).)

This list is in no particular order – there is no "best guess" – 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 '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 a list of 69 offerings, including <corf>, <corti>, <carate>, <ceroid>, <chert>, <choreoid>, <karate>, <keyword>, <scirrhoid> and <scurried>, but not, sadly, <caught>.

Real-word errors

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 supplements to the spellchecker's main dictionary. On the other hand, the spellchecker failed to flag real-word errors, where the error is a dictionary word but not the right one (<there> for <their>, and the like). 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 early approach to this problem (Atwell and Elliott, 1987) homed in on the grammatical anomalies that were often produced by real-word errors. For example, the sentence, 'Please complete the *from* in capitals,' would be flagged because the sequence Definite Article-Preposition-Preposition is very unlikely. It had some

success in spotting real-word errors but tended to raise too many false alarms, querying sentences where the syntax was unusual but not incorrect (Leech et al., 1986). And of course it could do nothing with such sentences as, 'We had thirty *minuets* for lunch,' where the anomaly is semantic, not syntactic.

Another early approach (Mays et al., 1991) used a table of word-sequence probabilities. Given any two words from its 20,000 word dictionary, the table would give the probability of any other occurring next. Suppose their spellchecker was checking 'The thief licked the lock.' Having consulted the table for the probability of <licked> after 'The thief', <the> after 'thief licked', and so on, it was able to calculate the probability of the whole sentence. It also calculated the probability of all the sentences that differed from the first by a single typo. In this case, it would try <liked>, <kicked>, <picked> and so on in place of <licked>, and also <rock>, <sock> etc in place of <lock>. If any of these variant sentences was significantly more probable than the original, the modified word (perhaps <picked> instead of <licked>) would be offered as a correction.

The next assault on real-word errors made use of confusion sets (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.' 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 and only to flag an error if the spellchecker is confident, 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.

The early research with this technique used a list of confusion sets that was both short (18 sets in (Golding, 1995)) and with only a small representation of function-word confusions (such as {<he>, <her>} or {<to>, < too>}) that account for a high proportion of genuine real-word errors (Mitton, 1996). Though the list was sufficient for proof of concept – and Golding makes no grander claim for it – this was unfortunate in retrospect since most of the subsequent experiments used the same list, to preserve comparability with earlier work, though (Carlson et al., 2001) scaled it up to 265. More recently an attempt has been made to produce a list sufficiently large to tackle unrestricted text (Pedler and Mitton, 2010), but experiments with a test corpus of errors collected from student essays, online bulletin boards and so on, suggest that, even with their list of about 6,000 confusion sets, around 30% of the real-word errors would remain undetected.

Might it not be possible to get a computer to detect real-word errors in the same way as humans do, by seeing that the errors don't make sense in the context? An attempt

to do this was implemented by (Hirst and Budanitsky, 2005). Take the sentence, 'We had thirty minuets for lunch.' Their system would check whether any of the meanings of <minuet> (dance, music) were close to the meanings of any other word in the paragraph. If none of them were, and if there was a word very similar to <minuets> (here <minutes>), and if this other word did have meanings that were close to the meanings of some others in the vicinity (perhaps <midday> or <timetable> occur elsewhere in the paragraph), it would propose <minutes> as a correction for <minuets>. It collected the meanings from WordNet (Miller, 1995; Fellbaum, 1998, 2005), a large lexical database that lists the meanings of around 100,000 English words and indicates the relationships between their meanings (synonyms, hyponyms etc.). The notion of closeness between the meanings of words in WordNet is a slippery one; there are several definitions and they have different results – see, for example (Budanitsky and Hirst, 2006).

Though this semantic approach showed some promise, it has been largely put aside in favour of experiments with the very large amounts of text that have become available by harvesting the world-wide web – see below under "Big data".

Ordering the list of suggestions: assembling candidates

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 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 set of candidate corrections; these are words that somewhat resemble the error – perhaps five to fifty of them, perhaps some hundreds, depending on the system. But how to pick out these candidates without trawling through the entire dictionary every time?

One way is to create a key for each word in the dictionary, which will retain the prominent features of the spelling, the features which you hope misspellings of that word are likely to retain. Faced with a misspelling, you compute a key for the misspelling and retrieve all the dictionary words that have the same key.

One such key, called Soundex (Odell and Russell, 1918, 1922), was devised decades before computers. It was invented to address the problems caused by names having variant spellings – a clerk might be searching for someone's record in a card index without being sure how their name is spelt, or someone might appear under one spelling in one file and under a different spelling in another. A Soundex key (for details see, for example, (Mitton, 1996)) retains the first letter of the name followed by a few digits representing the consonant structure. For example, <Johnson>, <Jonson> and <Johansson> all have the Soundex key J525. (The system is still used in genealogical searches; see, for example (US National Archives, 2007).)

Whereas Soundex was designed for human beings to use, more recent implementations of the same idea have been designed for computers and can therefore be more complicated. An example of this is Metaphone (Philips, 1990), and its

successors (Philips, 2000, 2009). They differ from Soundex in having more, and more detailed, rules; for example, if a word begins with <pn> (e.g. <pneumatic>), the computed key begins with N, not P.

The same basic idea underpins the "People Search" services on the web. In (Udupa and Kumar, 2010), for example, a key is created for each name in a directory of possibly millions of names. When the user types in a guess at a name, perhaps misspelled or in the wrong order or with parts missing (<Tim Berners>, <Tom Barnard Lea>, <Tim Lee Burners>, etc) a key is derived from the user's guess, and all the names are retrieved that have the same key, or something close to it.

These key systems in spellchecking mostly belong to a time when computer memories were too small to hold a complete dictionary, but capacity has increased to the point where it is possible to hold a complete dictionary in main memory, and data structures and algorithms have been developed to take advantage of this. It is now possible to retrieve all the words that differ in a well-defined way from the misspelling, perhaps words of the same length as the misspelling that differ in a certain number of character positions (Bentley and Sedgewick, 1997) or words that differ from the misspelling by a given edit-distance (Mihov and Schulz, 2004); the concept of edit-distance is explained below.

Ordering the list of suggestions: string-matching

One way or another, a list is generated of candidates that somewhat resemble the misspelling. 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 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.

You calculate the edit-distance for each of the candidates and then present them in order, lowest first. 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, 2008), 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 (as in, say, <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 table applicable to all words – you might decide 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; changing a <p> to an <f>, for example, 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.

The costs can even 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 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 . <Latest> is sometimes written <lastest>, so we attach a low cost to the insertion of the first <s>.

An improvement on these rather arbitrary costs is to use probabilities. The system described by (Kernighan at al., 1990), which was aimed at correcting errors that contained just one typo, used a table derived from millions of words of typewritten text, giving the probability that, say, an <e> would be substituted for an <o>, or a <t> omitted after a <c>. So, given a misspelling <acress>, it would use this table to assess the probability that the target was <across> or <actress>.

The system employed by (Brill and Moore, 2000) was similar except that it used word-fragments rather than single letters. Their table, derived from a large collection of misspellings, gave the probability of, for example, <ant> being written for <ent>. More precisely it gave three probabilities – one if the substitution occurred at the start of a word, one at the end and the third in between. It would use the word-final <ant/ent> probability in assessing whether <presant> was a likely misspelling of <present>.

A development of this system (Toutanova and Moore, 2002) used a similar table of fragments, but this time of pronunciations rather than spellings. Given a misspelling, it would make a guess at the pronunciation and compare this with pronunciations in the dictionary. For example, whereas the spelling-based system, described in the last paragraph, proposed <grizzle> for <grissel>, the pronunciation-based one correctly proposed <gristle>.

The "Cupertino"

Spellcheckers became 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. Whether from an excess of faith or a lack of attention, people sometimes choose 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'.

Big data

When describing the confusion-set approach to the correction of real-word errors (see above), I was rather vague about how exactly the spellchecker, having encountered an occurrence of, say, <principle>, decides whether <principal> would be more appropriate in that context. A technique often employed for this problem (and, incidentally, for a number of other problems in natural-language analysis) is to take a large corpus of text and to contrast the sort of contexts in which <principle> appears with those in which <principal> appears. Armed with the results of this analysis, the spellchecker tries to decide whether the current context resembles the <principle> sort or the <principal> sort.

Typically a researcher would divide the large corpus of text into a training corpus, which provided the basis for calculating the statistics that would underpin these decisions, and a test corpus on which experiments would be carried out to see how well the spellchecker corrected the errors (or left the correct uses untouched, as the case may be).

For many years the corpus of text on which these analyses were based was the Brown Corpus (Kucera and Francis, 1967), a million words of American English text from the early 1960's, sampled to contain representations of various genres. Given the corpus-building technology of the time, a million-word corpus was large, and subsequent corpora tended to be about this size, such as a British-English version (Leech et al., 1976). Effort went into enhancing the corpora, for example by tagging each word with its part-of-speech, rather than making them larger, though a corpus based on the Wall Street Journal (Charniak at al., 1989) reached thirty million words and the British National Corpus (Burnard, 1995) reached one hundred million.

In real-word error checking, researchers had tended to concentrate on the types of statistics they derived from the training set and the ways they used them in deciding between confusion-set members. But it was pointed out (Banko and Brill, 2001) that there was more progress to be made simply from using larger training sets, and that larger (much larger) corpora could now be obtained by harvesting text from the world-wide web. By using error-checking techniques that had already been developed, but training them on corpora of various sizes between one million and a billion words, they showed that performance got better and better with the size of the training set, regardless of the techniques used, and showed no sign of levelling off even at a billion. In the oft-quoted words of the American researcher R.L. Mercer – see (Jelinek, 2004), "There's no data like more data."

Over the next decade, many experiments were carried out, taking advantage of very large corpora. One of these (Strohmaier et al, 2003) set out to improve the correction of output from an optical character reader (OCR). These machines are prone to make confusions at the character level, such as misreading <rn> as <m>, which obviously result in misspelt words, so they are equipped with spellcheckers that attempt to correct their output. The authors pointed out that a text tends to be about a particular topic; the OCR's dictionary, even if it is a large one, will be a general one, possibly weak on the specialist vocabulary of the topic in question. They selected six topics and put a test text on each through an OCR, which they endeavoured to correct using either a large, general dictionary or a specialised dictionary which they created by putting queries with specialist terms to a search engine and then harvesting words from web pages, using the suggested sites as starting points and following links – "web crawling". For each of the test texts, the OCR's correction was better with the web-crawled dictionary than with the large, static one.

The use of large corpora for spellchecking was given a boost by the publication in 2006 of the Google n-grams (Brants and Franz, 2006). This was a collection of word sequences, each with its frequency, from a corpus of around one trillion words of text taken from the web. The n-grams vary in length from unigrams (single words) to five-grams; for example, the three-grams contain "ceramics collected by : 52" and "ceramics consist of : 92"; the four-grams include "serve as the independent : 794" and "serve as the initial : 5331". Words that occurred less than 200 times are replaced by <UNK> and n-grams that occurred less than 40 times are not included.

One experiment to use this resource was (Carlson and Fette, 2007). Suppose their spellchecker came across 'The sand eel is the principle food for many birds.' Using the familiar list of 18 confusion sets (Golding, 1995), which included {<principle>, <principal>}, it would check whether <principle> was correct. It would collect the frequencies of 'sand eel is the principle', 'eel is the principle food', 'is the principle food for', 'the principle food for many', and 'principle food for many birds'. Then it would do the same for <principal>. After comparing the counts for <principle> versus <principal>, it would decide whether to accept <principle> or to suggest <principal> instead. If the five-grams proved unhelpful, it would try four-grams, and so on, down to unigrams if necessary (in which case it would simply be choosing the more frequent of the two words).

N-grams can also be used in the correction of non-word errors (Carlson and Fette, 2007; Flor, 2012). Once a list of candidate corrections has been assembled, they can

be scored on how frequently they appear in exactly the same context as the misspelling, and the list reordered accordingly.

A particularly striking use of the web (Whitelaw et al, 2009) implements spellchecking without any dictionary at all. They begin by taking a sample of over a billion web pages of English text, listing all the words that occur in them and retaining the most frequent ten million. This will include most of the correctly spelled words of English, but a large number of misspellings also (and other odd items such as numbers, names and web addresses).

To make a guess at which words are misspellings, and of which target words, they first take each word in turn and pair it up with all the other words that differ from it by the insertion, deletion or substitution of just one letter (they make no attempt at misspellings that differ from their target by more than this). If one member of a pair is at least ten times more frequent than the other, the less frequent is considered to be a possible misspelling of the more frequent. They then consider all the three-word contexts in which either of the words occurs. Where the contexts are the same, they guess that the less frequent is a misspelling of the more frequent, in that context. Where the contexts differ, they consider them to be distinct words.

For example, the non-word <accidential> is paired with <accidental>, the latter being much the more frequent. The majority of the contexts are the same, so <accidential> is taken to be, in most of its occurrences, a misspelling of <accidental>. The system can also spot real-word errors. <Occidental>, for example, is also paired with <accidental>, and, where the contexts are the same, it is taken to be a misspelling of <accidental> (a real-word error). But most of the contexts are different, and in all those contexts it is taken to be a separate word.

At the end of this process, they have over 100 million triples of the form: (C, M, n) where C is the (assumed to be) correct spelling, M is a (probable) misspelling, and n is the number of times they appeared in the same context. This dataset is far from perfect – there are many obvious misspellings that are not paired with their correction – but it provides a large training set from which the system proceeds to extract statistics on how likely it is that some word-fragment x in a misspelling corresponds to a fragment y in a correction. When the spellchecker checks a text, these statistics form the basis for putting into order of likelihood the list of suggestions for a misspelling.

There is insufficient space here to describe the further steps that go into the production of this spellchecker. Suffice it to say that, despite the amount of noise in the original data and the level of guesswork that goes into the spellchecker's creation, not to mention the complete absence of any human checking or correcting, the performance of the final product bears comparison to that of Aspell, a well-known open-source spellchecker (Atkinson, 2011). And this has been achieved without any dictionary or any other hand-crafted data source, such as annotated lists of selected misspellings. For that reason, the system can be applied to produce spellcheckers for other languages, and the authors demonstrate this by producing versions for German, Russian and Arabic.

Spellchecking search-engine queries

When I began my research into spellchecking in the 1980's, I gave a presentation on my ideas to my academic colleagues, 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. Thirty 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, 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. Out of a hundred secondary-school attempts at <scissors>, there will be perhaps a dozen <sissors>; there will also be lots of wilder variations, such as <cezzous>, <saciarres>, <sisions> or <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)

The future

Whether these big-data techniques can be transferred to your own computer depends on the future of computing. You certainly could not accommodate gigantic files of downloaded text on your laptop. 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 – an arrangement known as "cloud computing" – so that the spellchecking of your documents, like the spellchecking of your search-engine queries, 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, it may be that you will be benefiting not so much from the efforts of good spellers who have gone before you, patiently creating dictionaries of correct spellings, but from the efforts of bad ones, misspelling the same word in a thousand different ways.

References

Atkinson, K. (2011). *GNU Aspell*. aspell.net.

Atwell, E. and Elliott, S. (1987) Dealing with ill-formed English text. In Garside, Leech and Sampson (eds) *The Computational Analysis of English: a corpus-based approach*. Longman, pp. 120-138.

Banco, M. and Brill, E. (2001).Scaling to very very large corpora for natural language disambiguation. *ACL '01: Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics:* 26-33.

Bentley, J. and Sedgewick, R. (1997). Fast algorithms for sorting and searching strings. *SODA '97: Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*: 360-369.

Brants, T., and Franz, A. (2006). *Web 1T 5-gram Version 1*, Linguistic Data Consortium, Philadelphia.

Brill, E. and Moore, R.C. (2000). An improved error model for noisy channel spelling correction. *ACL '00 Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics,* 286-293.

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.

Budanitsky, A. and Hirst, G. (2006). Evaluating WordNet-based measures of lexical semantic relatedness. *Computational Linguistics* **32** (1): 13-47.

Burnard, L. (1995). Ed. *Users' Reference Guide for the British National Corpus version 1.0,* Oxford: Oxford University Computing Services.

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.

Carlson, A.J. and Fette, I. (2007). Memory-based context-sensitive spelling correction at web scale. *ICMLA 2007 Sixth International Conference on Machine Learning and Applications*, 166-171.

Charniak, E., Blaheta, D., Ge, N., Hall, K., Hale, J., Johnson, M. (1989). BLLIP 1987-89 WSJ Corpus Release 1, Linguistic Data Consortium, Philadelphia.
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**: 171-176.
Damerau, F.J. and Mays, E. (1989). An examination of undetected typing errors. *Information Processing and Management*, **25** (6): 659-664.
Fellbaum, C. (1998). *WordNet: an Electronic Lexical Database*. Cambridge MA: MIT Press.
Fellbaum, C. (2005). WordNet and wordnets, in Brown, Keith et al (eds) *Encyclopedia of Language and Linguistics*. Oxford : Elsevier, 665-670.
Flor, M. (2012). Four types of context for automatic spelling correction. *Traitement Automatique des Langues*, **53** (3): 61-99.
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**: 107-130.
Hirst, G. and Budanitsky, A. (2005). Correcting real-word spelling errors by restoring lexical cohesion. *Natural Language Engineering* **11** (1): 87-111.
Jelinek, F. (2004). Some of my best friends are linguists, *Language Resources and Evaluation Conference 2004* presentation.
Jurafsky, D. and Martin, J.M. (2000). *Speech and Language Processing*. New Jersey: Prentice Hall.
Kernighan, M.D., Church, K.W. and Gale, W.A. (1990) A spelling correction program based on a noisy channel model. In Karlgren, H. (ed.) *COLING-90 13th International Conference on Computational Linguistics.* Helsinki. **2**: 205-210.
Kucera, H. and Francis, W.N. (1967) *The Computational Analysis of Present-Day American English*, Brown University Press.
Kukich, K. (1992). Techniques for automatically correcting words in text. *Computing Surveys*, **24** (4): 377-439.
Leech, G., Johansson, S., Hofland, K. (1976). *The LOB (Lancaster-Oslo-Bergen) Corpus.*
Leech, G.N., Garside, R.G. and Elliott, S.J. (1986) *Development of a Context-sensitive Textual Error Detector and Corrector: Final project report submitted to International Computers Limited.* Unit for Computer Research on the English Language, Lancaster University.
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): 707-710.
Mays, E., Damerau, F.J. and Mercer, R.L. (1991). Context based spelling correction. *Information Processing and Management*, **27** (5): 517-522.
McIlroy, M.D. (1982). Development of a spelling list. *IEEE Transactions on Communications*, **COM-30** (1): 91-99.
Mihov, S. and Schulz, K.U. (2004). Fast approximate search in large dictionaries. *Computational Linguistics*, **30** (4): 451-477.

Miller, G. A. (1995). WordNet: a lexical database for English. *Communications of the ACM,* **38** (11): 39-41.
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): 173-192.
Mitton, R. (2010). Fifty years of spellchecking. *Writing Systems Research*, **2** (1): 1-7.
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.
Odell, M.K. and Russell R.C. (1918, 1922). U.S.Patents 1261167, 1435663.
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): 676-687.
Peterson, J.L. (1986). A note on undetected typing errors. *Communications of the A.C.M.*, **29** (7): 633-637.
Philips, L. (1990). Hanging on the Metaphone, *Computer Language*, **7** (12): 38-44
Philips, L. (2000). The Double Metaphone search algorithm, *C/C++ Users Journal*
Philips, L. (2009). Metaphone 3, software product
Pollock, J.L. and Zamora, A. (1984). Automatic spelling correction in scientific and scholarly text. *Communications of the A.C.M.*, **27** (4): 358-368.
Sterling, C.M. (1983). Spelling errors in context. *British Journal of Psychology*, **74**: 353-364.
Strohmaier, C.M., Ringlstetter, C., Schulz, K.U., Mihov, S. (2003). Lexical postcorrection of OCR-results: the web as a dynamic secondary dictionary? *ICDAR*, IEEE Computer Society, 1133-1137.
Toutanova, K. and Moore, R.C. (2002). Pronunciation modeling for improved spelling correction. *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*: 144-151.
Udupa, R. and Kumar, S. (2010). Hashing-based approaches to spelling correction of personal names, *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing (EMNLP2010)*: 1256-1265.
U.S.National Archives (2007). www.archives.gov/research/census/soundex.html
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): 168-173.
Whitelaw, C., Hutchinson, B., Chung, G.Y., Ellis, G. (2009). Using the web for language independent spellchecking and autocorrection. *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, 890-899.

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, 251-285.