Spell Checking using the Google Web API
Su Zhang
University of Kent
sz24@kent.ac.uk

Abstract
Spellchecking systems are commonly used in document preparation. They can be categorized as misspelling identifiers and correctors, or one spellchecking system has both these two functions. However, they have some obvious drawbacks; for instance, the identifiers are not able to catch real world errors. The correctors often provide users with unusual and inappropriate suggestions. As most of them make little use of in-sentence context. Recently, some experimental systems attempt to use context, but they use a limited model of sentence context due to the requirement to store the text-base on the user’s machine. This paper investigates the state-of-art in spellchecking research and evaluates the accuracy and usability of a new spellchecking program named Google spell checker that is able to improve the performance of detecting and correcting real-word errors.

1. Aims
The aims of this project are to implement a new spellchecking program named Google spell checker that applies the Google web API to get the text on the fly and searches for occurrence of n-word fragments in real context. Furthermore, is to write a technical report for investigating the state of art in spell checking, and demonstrating and evaluating the new spell checker.

2. Introduction
This paper first introduces the background of spellchecking. In this section, different types of spelling errors such as non-word and real-word errors will be introduced. Then it explicitly presents several most commonly used misspelling detection techniques. Last but not least, it gives several spellchecking systems as examples that apply different techniques for misspelling corrections. Secondly, within the technical section, it introduces the initial Google spell checker and the current one, demonstrates different techniques they have applied, and discusses the advantages and disadvantages they have. Then it mainly evaluates the spell checker’s performance for handling real-word errors, and lastly, it gives a quick comparison between the spell checker and other non-word error based systems.

3. Background
3.1 Spelling Errors
Nowadays, there are two distinct types of spelling errors, namely, non-word error and real-word error. And Damerau [3] indicates that 80 percent of misspellings are categorized by four rules, there are transposition of two letters, one letter extra, one letter missing and on letter wrong (Transposition, insertion, deletion and reverse).

3.1.1 Non-word Error
The definition of non-word is that a word does not have a meaning. According to Kukich [8], normally, typed text has three types of non-word error, namely, (1) typographic errors, (2) cognitive errors and (3) phonetic errors. In the case of typographic errors, it is assumed that the typist knows the correct spelling but presses the wrong key by accident, and or presses the keys in wrong order. e.g. and → and
In the case of cognitive errors, it is assumed that the typist does not have the knowledge or misunderstands the intended word. e.g. minute → minite
In the case of phonetic errors it is assumed that the typist substitutes a phonetically correct but orthographically incorrect sequence of letters for the intended word. e.g. two → too, their → there

3.1.1 Real-Word Error
According to Mitton [13], “real-word error is a valid word but not the intended word in the sentence, thus making the sentence syntactically or semantically ill formed or incorrectly.” e.g. there → their. A study was mentioned by Kukich [8], real-word errors are classified in three main categories: (1) real-word errors in locally invalid syntactic contexts; (2) real-word errors in globally invalid syntactic contexts; and (3) syntactically valid real-word errors in invalid semantic contexts. There are some significant findings to show real-word error are ubiquitous, Mitton [12] reported that 40% of the misspelling generated from 925 handwritten student essays are real-word errors, and Wing and Baddeley [18] indicated that 30% of 1,185 handwritten errors involve real-word errors.
3.2 Error Detection and Correction

3.2.1 N-gram Analysis Techniques

N-gram analysis is one of the popular methods for detecting non-word errors. Normally, it is used to detect errors made by Optical Character Recognizers (OCR) devices. N-grams are n letters subsequences of words or strings. N stands for one, two or three. One-letter n-grams are referred to as unigrams or monograms; two-letter n-grams are referred to as bigrams; and three-letter n-grams are seen as trigrams.

In order to pre-compile an n-gram table, a dictionary or corpus of text is usually required. The table stores n-gram’s existence or frequency, any n-grams in an input string that have nonexistence or low frequencies are classified as probable misspellings. The table has a variety of forms, such as binary bigram array or binary trigram array. In the case of binary bigram array, it has two-dimensional array of size 26 * 26 whose elements represent all possible bigrams. For each element, if it occurs in at least one word (string) in predefined dictionary or text, then its value set to 1, otherwise set to 0. A binary trigram array could have three-dimensional array. Since these binary n-grams do not indicate the position of each n-gram within each word, they are nonpositional binary n-gram arrays. Moreover, it is said that a set of positional binary n-gram arrays are able to detect error more accurately. Because each element in the positional binary n-gram arrays matches the exact position within each word. However, this raises the storage space problem due to the large capacity of the positional arrays. Since most misspellings do not contain any impossible n-grams, so n-gram analysis techniques are not good at detecting human-generated errors but good at detecting machine-generated errors.

3.2.2 Dictionary Lookup Techniques

Dictionary lookup is another common technique for detecting non-word error. Because it is a simple and quick task, it just simply lookup each word in a dictionary; any unfound words are considered as misspellings. The basic algorithm for spell checker to use dictionary lookup is:

**Initialize**

**Build list.** Construct a list of all distinct tokens in the input file. (See the algorithm for constructing the distinct token list)

**Search.** Look up each token of the list in the dictionary.

- If the token is in the dictionary, it is correctly spelled.
- If the token is not in the dictionary, it is not known to be correctly spelled and is put on the output list.

**Print.** Print the list of tokens that were not found in the dictionary.

And the basic algorithm for constructing a list of distinct tokens:

**Initialize.** Get the name of the input file and the output file, open them, and set all variables to their initial state.

**Loop.** While there are still tokens in the input file, get the next token and enter it into an internal table of tokens.

- To enter the token, first search to see if it is already in the table. If not, add it to the table.

**Print.** When all tokens have been read, print the table and stop.

However, response time and storage space raise issues when the size of the dictionary increases. In order to improve the access speed to a dictionary, several standard search techniques have been introduced, such as hash table [6], tries [6], frequency-ordered binary search trees [6], finite-state automata [1] and median split trees [15], and hash table is the most commonly used approach. It computes each input string’s hash address and stores it at the address. To lookup the input string, if it is different from the retrieved word stored at the table or is null, then it is misspelling. Turba [16] outlined that the main advantage and disadvantage of using hash tables; the nature fast random access of hash code avoids the large number of comparisons needed for tree-based searches or sequential of the dictionary. However, an efficient hash function has to be devised to avoid collisions occurred in construction of the hash tables.

There is alternative approach suggested by Peterson [13], a dictionary can be partitioned into three separate dictionaries for spelling correction. The first one has a few hundred most commonly used English words that stored in the cache memory, it accounts for 50% of the complete dictionary access. The second one consists of a few thousand document specific words that stored in the regular memory, it accounts for 45% of the access. The third one accounts for the remaining 5% of the access, has complete dictionary (probably stored on disk) that stored in the second memory.

The size of a dictionary is another issue,
Kukich [8] stated that too small a dictionary produces too many false alarms of valid terms; too large a dictionary led to an unacceptably high number of false acceptance. Furthermore, Peterson [13] said that a large dictionary often includes rare, archaic, or obsolete words that tend to result in errors in technical writing. Peterson [13] suggested creating multiple dictionaries to limit the size of dictionary; each local dictionary is for each specific-domain. To check a particular file, several selected local dictionaries maintained by administrator and individual user to create a temporary master dictionary.

Saving space is an essential factor of spell checkers, McIlroy [10] introduced the ‘affix-stripping’ which only holds the stems of words, iteratively removes suffixes or prefixes until it reaches a stem, or iteratively removes suffixes and prefixes at the same time to reach the stem. But this method would still accept some non-word errors. Moreover, Nix [11] stated that holding dictionary as a bitmap can improve the saving space problem, it goes through the dictionary and converts each word into a number (hashing), set all the converted words to 1. To do spell checking, it converts each word of the input text to a number and compares this number to the appropriate address, if they are the same, then it is correct spelling, otherwise it is spelling error. The basic principles of this method is that using a spell checker, a user can check a particular file, several selected local dictionaries maintained by administrator and individual user to create a temporary master dictionary.

3.2.3 NLP (Natural Language Processing) Prototypes For Handling Ill-formed Input

Nowadays, context-dependent word detection and correction tools have natural-language-processing (NLP) capabilities, including robust natural language parsing, semantic understanding, pragmatic modeling, and discourse structure modeling. Most prototype NLP systems are parser driven and incorporate with error-handling techniques. According to Kukich [8], “A typical NLP system consists of a lexicon, a grammar, and a parsing procedure. The lexicon is the set of terms that are relevant to the application domain. The terms are usually annotated with their potential parts of speech as well as morphological information. The grammar is a set of rules that specifies how words, parts of speech, and higher-order syntactic structures may be validly organized into well-formed sentence fragments and sentences. The parsing procedure often consists of looking up each word in the dictionary to determine its potential parts of speech and applying grammar rules to build higher-order syntactic structures. Since many words” Three general approaches to deal with ill formed input, there are (1) Acceptance-based techniques, (2) Relaxation-based techniques, and (3) Expectation-based techniques. Recently, several NLP systems have been devised with these rule-based techniques to handle real-word errors. There are two relaxation-based text-editing systems, the EPILSTLE/CRITIQUE system and a text editor for the Dutch language, all of them are mainly focus on syntactic processing, and therefore, they are most well suited for use on unrestricted text. And Kukich [8] says, “In relaxation-based techniques, when a parsing failure occurs, the system attempts to locate an error by identifying a rule that might have been violated and by determining whether its relaxation might lead to a successful parse.”

Case study 1

IBM develops the EPILSTLE/CRITIQUE system for editing business correspondence (Heidorn et al. 1982). It detects grammatical anomalies. It is parser-based writing aid tool, it preprocesses the input to detect and correct nonword errors before the parsing procedure; certain real-word errors are detected during the parsing process. It holds a complicated set of grammar rules for English; it uses these rules to parse each sentence of the input text. But Kukich [8] indicated that EPILSTLE/CRITIQUE was originally designed to diagnose five classes of grammatical errors: (1) subject-verb disagreement, (2) wrong pronoun case, (3) noun-modifier disagreement, (4) nonstandard verb forms, and (5) nonparallel structures. If it encounters a grammatically incorrect sentence, it tries to continue relaxing some of its grammar rules to parse the sentence until it reaches a successful paring. Since it knows which rule or rules it has relaxed, the grammatically incorrect part(s) of the sentence can be detected and corrected with suggestion(s).

Case study 2

The text editor for the Dutch language uses Van Berkel and DeSmedt spelling corrector [17] and a unification-based parser consisting
of around 500 augmented phrase structure rules. It has a 250,000-word lexicon that is produced from a 100,000-word dictionary, which is cross-indexed to inflected forms. It processes a document in four steps: (1) preprocessing, (2) word-level analysis, (3) sentence-level analysis, and (4) text resynthesis. Firstly, it removes the markup symbols and corrects punctuation. Secondly, it uses the Van Berkel and DeSmedt spelling corrector to generate a small list of candidate corrections for each word in the document. Thirdly, it uses unification-based parser to perform high-level parse. If syntactic violations are detected, unification rules are relaxed to construct a valid parse using candidates from the candidate correction set. Lastly, it produces a new document with corrected errors and the error messages.

3.2.4 Statistically Based Error Detection and Correction

The statistical language-modeling approach is an alternative for real-word error correction. “Statistical language models (SLMs) are essential tables of conditional probability estimates for some or all words in a language that specify a word’s likelihood to occur within the context of other words.” mentioned by Kukich [8]. Recently, there are two studies using various SLMs-based to detect or correct real-word errors, each study uses one of these two models, Part-Of-speech (POS) bigram model, or a word trigram model.

3.2.4.1 Part Of Speech (POS) Bigram Probabilities To Detect Real-Word Errors

A statistical language modeling based application is developed at the University of Lancaster’s Unit for Computer Research on the English Language (UCREL) [4] This system tags words in a text within their part of speech. A probabilistic word tagger called CLAWS (Constituent Likelihood Automatic Word-tagging System) enables it to tag words. Kukich [8] pointed out that CLAWS uses a syntactic-tag set consisting of 133 part of speech categories and a part of speech bigram SLM. A tagged corpus of one million words estimates the POS transition probabilities. CLAWS determines which tag or tags each word in a sentence can have. Normally, approximate 65% of all words only have one tag. And the system uses the highest POS bigram transition probability to solve the remaining 35%. And as Kukich [8] stated, “This is accomplished by computing the path of maximum probability for a sequence of one or more ambiguously tagged words bounded at either end by unambiguously tagged words.”

3.2.4.2 Lexical Trigram Probabilities To Detect And Correct Real-Word Errors

Mays et al [9] introduced his study that employs a word trigram SLM based on a 20,000-word lexicon and uses a set of 100 sentences containing only words found in the lexicon. It uses probabilities derived from a very large corpus of text, and the probabilities are of co-occurrence of actual words. It is able to say what the probability is of any other of the 20,000-word lexicon words follow any two words chosen from the lexicon. For example, ‘You think’ is the first two words, ‘that’ has relatively high probability after these first two words, while, the probability of ‘big’ after the two words is almost zero.

An experiment is reported by Roger [16], it tested the Mays et al’s study on three-word sentences contain a single real-word error. The study generates all possible candidate corrections that altered from the possible error; it calculates the probability of the whole sentence from its table of three-word probabilities and calculates the probability of the original sentence. Finally, it compares each candidate’s probability to the original’s probability. If the candidate’s probability is higher, then the possible error in the original should be corrected to the corresponded one within the candidate.

3.2.5 Error correction systems

It is said that people are good at detecting errors, but are not good at correcting them. However, it is not easy for computers to spot out the errors, but once the errors are detected, they can easily correct them. Since computers have difficulty deciding intended words of a sentence. Moreover, computers can detect a minor misspelling in a long word. By contrast, they might not detect a misspelling in a short word. In what follows, several spelling correction systems will be introduced.

3.2.5.1 Soundex

Originally, the Soundex system is employed for retrieving customers’ details from Database systems [7]. It creates a Soundex code for every name in a Database system. It ignores vowel letters and groups consonant letters that have high probability of substituting for each
This technique can be applied to spelling correction; it creates a Soundex code for every word in a dictionary and a Soundex code for the misspelling. Then it retrieves those words from the dictionary have the same code as the misspelling.

And the basic algorithm is:
1) Keep the first letter of the word (in upper case).
2) Replace these letters with hyphens: a, e, i, o, u, y, h, w.
3) Replace the other letters by numbers as follows:
   \[
   \begin{align*}
   b, f, p, v & : 1 \\
   c, g, j, k, q, s, x, z & : 2 \\
   d, t & : 3 \\
   l & : 4 \\
   m, n & : 5 \\
   r & : 6
   \end{align*}
   \]
4) Delete adjacent repeats of a number.
5) Delete the hyphens.
6) Keep the first three numbers or pad out with zeros.

For instance:

<table>
<thead>
<tr>
<th>toy</th>
<th>Birkbeck</th>
</tr>
</thead>
<tbody>
<tr>
<td>Toy</td>
<td>B-621-22</td>
</tr>
<tr>
<td>T000</td>
<td>B-621</td>
</tr>
</tbody>
</table>

### 3.2.5.2 SPEEDCOP

The SPEEDCOP system is devised to correct typing errors. The core of the system is correcting isolated error that contains only single error and whose correct forms in a dictionary. In order to correct other types of errors, the system uses a misspelling dictionary and a function word routine [14].

The basic SPEEDCOP correction algorithm:
1) Generate a similarity key for each word in the dictionary.
2) Sort the dictionary in key order.
3) Generate a key for the misspelling.
4) Locate words whose keys collate most closely to the key of the misspelling.
5) Select the plausible correction(s) from these candidate words.

Two properties of an alphabetic string are the identity and interrelationship of the letters that comprise it [14]. The system uses two keys, (1) Skeleton key and (2) Omission key. The first key is constructed by concatenating the following features of the string (word or misspelling), first letter + consonant letters that ordered by their occurrence in the word + vowel letters that ordered by occurrence. Each of these letters recorded only once.

It is true that most failures of skeleton key correction are caused by omission errors. Therefore, the omission key is employed to solve this problem. It omits consonant from words in the following frequency order: RSTNLGDPMFBYWVZXQJK [14]. In this order, R is omitted more often than any other letters, while, J is less often. And Pollock and Zamora [14] suggested that the omission key of a particular string is constructed by sorting its unique consonants in the reverse of the frequency order and appending its unique vowels in their original order.

And these two types of similarity keys are tested on different sources of data, ACS (American Chemical Society), CA (Chemical Abstracts), BA (Biological Abstracts), CIN (Chemical Industry Notes), ISA (Information Science Abstracts) and PI (Philosopher’s Index).

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Corrected (%)</th>
<th>Miscorrected (%)</th>
<th>Uncorrected (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>S-KEY</td>
<td>O-KEY</td>
<td>S-KEY</td>
</tr>
<tr>
<td>ACS</td>
<td>65.81</td>
<td>71.00</td>
<td>10.85</td>
</tr>
<tr>
<td>ACS-2</td>
<td>68.51</td>
<td>74.40</td>
<td>9.20</td>
</tr>
<tr>
<td>BA</td>
<td>61.04</td>
<td>66.87</td>
<td>5.25</td>
</tr>
<tr>
<td>CA</td>
<td>71.73</td>
<td>76.95</td>
<td>12.36</td>
</tr>
<tr>
<td>CIN</td>
<td>57.61</td>
<td>61.89</td>
<td>11.42</td>
</tr>
<tr>
<td>DOLE</td>
<td>58.73</td>
<td>65.16</td>
<td>11.10</td>
</tr>
<tr>
<td>ISA</td>
<td>54.02</td>
<td>57.34</td>
<td>11.91</td>
</tr>
<tr>
<td>PI</td>
<td>48.75</td>
<td>56.25</td>
<td>3.75</td>
</tr>
</tbody>
</table>

Table 1. Cumulative effectiveness of the similarity keys

### 3.2.5.3 Plausibility-based correction

Error-reversal technique detects misspellings depending on whether a misspelling can be transformed into a dictionary word by reversing one of the basic error operations (Transposition, insertion, deletion, reverse).

And the algorithm for this technique is:

Find the leftmost position \( P \) at which the strings differ.

IF the (potential target) word is longer than the
misspelling AND the word from
the position \( P + 1 \) to the end is identical to the
misspelling from position \( P \) to the end.
THEN the misspelling and the word differ
by a DELETION error.
ELSE IF the word is shorter than the
misspelling AND the word from position
\( P \) to the end is identical to the misspelling
from the position \( P + 1 \) to the end.
THEN the misspelling and the word differ
by an INSERTION error.
5. ELSE IF the two strings are the same
length
THEN
IF the two strings are identical from
position \( P \) on
THEN the misspelling and the word differ
by a SUBSTITUTION error.
ELSE IF a substring consisting of the \( (P + 1) \) th character of the misspelling followed by the
Pth character of the misspelling followed
by that part of the misspelling to the right of
the \( (P + 1) \) th character is identical to the target
word from the \( P \) th character to the end.
ELSE the word is not a potential correction

3.2.5.4 CORRECT

The CORRECT program is devised by Kernighan [5], which applies probabilistic techniques. It inputs a list of misspellings rejected by Unix spell, then proposes a list of candidate corrections for the errors and sorts them by probability.

In the first place, correct generates the candidate corrections by using the four basic error types, (insertion, deletion, substitution and transposition). In the second place, for each candidate correction, \( c \), correct calculates the probability score of \( c \) by applying the rule

\[
Pr(t|c) \approx \frac{\text{freq}(c)}{N} \times \frac{\text{freq}(c) + 0.5}{N}
\]

where \( \text{freq}(c) \) is the number of times that the word \( c \) appears in the 1988 AP corpus (\( N = 44 \) million words), this estimates the prior \( Pr(c) \). Moreover, the conditional probability, \( Pr(t|c) \), is constructed by four confusion matrices, (1) insertion\([x,y]\], the number of times that character \( x \) is typed as \( xy \), (2) deletion\([x,y]\], the number of times that characters \( xy \) are typed as \( x \), (3) substitution\([x,y]\], the number of times that characters \( x \) are typed as \( y \), (4) transposition\([x,y]\], the number of times that characters are typed as \( yx \). These matrices are divided by \( \text{chars}[x] \) and \( \text{chars}[x,y] \), the number of times \( x \) and \( xy \) appear in the training set to estimate the probabilities.

**Form the example above, the prior of each candidate correction is estimated by \( \text{freq}(c) + 0.5 \), and the prior is multiplies by the \( Pr(t|c) \) to get a raw score. In the case, “actress” has the highest score, so it is identified as the correct suggestion.**

4. Technical content

In this section, I would like to go in details what I have done. This section includes a comparison of different approaches that I considered, results obtained, problems encountered, performance measures, comparisons between different approaches adopted, comparisons with existing work on similar problems.

4.1 Web service

According to Chappel [2] “A web service is a piece of business logic, located somewhere on the Internet, that is accessible through standard-based Internet protocols such as HTTP or SMTP. A web service can have several technologies, namely, Simple Object Access Protocol (SOAP), Web Service Description Language (WSDL), and Universal Description, Discovery, and Integration (UDDI). In the case of SOAP, it is XML-based, extensible message envelope format, with bindings to underlying protocols. WSDL is also XML-format that describes the interface of a web service in a standardized way. Lastly,
UDDI is a protocol for publishing and discovering metadata about web services, it allows applications to find web services. In figure 1, the service requester acts as an application, the service broker is an web service publisher and finder, and the service provider is seen as web service holder, e.g. Google. In this scenario, the requester and the provider queries the broker for the service either by name, category, identifier, or specification supported. Then only the requester receives a WSDL document from the broker. And the requester creates and sends a SOAP message to the provider for requiring web service. Lastly, the provider sends the SOAP message back to the requester.

![Figure 1. Adapted from Wikipedia](image)

### 4.1.2 Simple Object Access Protocol (SOAP)

SOAP is a protocol for exchanging XML-based messages over a number of underlying protocols most commonly HTTP. It constructs the fundamental layer for the web services stack, which allows more abstract layers to be built on.

A SOAP message is a standard XML document that contains several specific elements. “Envelop” is the primary element for defining the SOAP document. And the “Envelop” element is constructed by three children. Firstly, the “Header” element is an optional element that contains specific information, for instance, authentication. Secondly, the “Body” element is compulsory, which defines the content of the SOAP message. Lastly, the “Fault” element is a child of the “Body” element that holds errors and status information for the content of the SOAP message.

![Figure 2. SOAP message structure](image)

### 4.1.3 Google SOAP Search API

Google SOAP search API is a web service provided by Google. It’s Google’s public interface for registered developers. It allows the developers to create and send/receive SOAP messages to the service provider.

### 4.2 Initial Google spell checker

The initial work is only capable of correcting single error within an input phrase. To what follows, I would like to present the fundamental approaches I have considered and the problems I encountered.

#### 4.2.1 Core technique: N-word fragment techniques

This has the similar principles to n-gram analysis techniques for detecting non-word errors. Instead of dividing a word into n-grams, it divides an input string into n-word fragments. Next, the new program is able to use the Web as a text repository to do context-dependent spell checking. It simply searches for the occurrence of each n-word fragment on the Google, and then retrieves each fragment’s number of returned results.

#### 4.2.2 Combination of the three types of word fragments

To detect bad fragments, a threshold is given to each type of word fragment. In this case, 2-word fragment’s threshold is set to 100,000, because the 2-word fragment has less contextual information than the other two, and the bad 2-word fragment attempt to large number of retrieval results. However, the other two types of word fragments’ thresholds are both set to 1000, because it has significant small of returned results if error exits in one of these fragments. Then it combines each type of n-word fragment’s generated bad fragments and good fragments to detect an error.

#### 4.2.3 Error detection method

This method applies the bad and good fragments generated from the combined n-word fragments mentioned above. The fragments here are the core technique to detect an error in the input string. The basic principle is that counting occurrence of each token of an input string then outputting
the token that has the most number of occurrences.

And the basic algorithm is:

1) Tokenize the input string, initialize a hash table, bad and good fragment list
2) For each fragment in the bad fragment list:
   For each token in the input string:
   If the bad fragment contains the token then
   If the token is not in the hash table then set the number of occurrence of the token to 0 and store both the token and its occurrence to the table
   Else increase the number of occurrence of the token by 1
   For each fragment in the good fragment list:
   For each token in the input string:
   If the good fragment contains the token then
   If the token is not in the hash table then set the number of occurrence of the token to 0 and store both the token and its occurrence to the table
   Else decrease the occurrence of the token by 1
3) Output the error from the table by comparing their number of occurrences

E.g. “The chat sat on the mat” is an input phrase
Bad fragments: The chat sat, the chat sat on, chat sat
Good fragments: sat on the, the mat, sat on the mat
Tokens: The, chat, sat, on, the, mat
Number of occurrences for each token: the (1), chat (3), sat (1), on (-1), the (-3), mat (-2)
The token has the most number of occurrences is identified as the misspelling; in this case, “chat” is flagged as the error.

There is a high probability that more than one word may have the same number of occurrence, so these words are stored in a bad word list.

4.2.4 The worst word detection method

This approach presumes that there is more than one word in the list; it first finds the worst 3-word fragment that has the least number of returned results from the bad fragment list, and then uses the bad word list and the worst fragment to identify the worst word. This uses similar approach as the error detection method.

4.2.5 Error correction method

There are four methods for four basic error types, namely, insertion, deletion, substitution, and reverse. In principle, these methods first generate a list of modification words for the error based on the four error types. Then it applies the simple dictionary lookup technique; it searches each modification word in a 41,000-word list to check if the word appears in the list. Finally, it builds a list of candidate words that are valid words.
The worst word and the worst 3-word fragment are essential components for correcting an error. And the basic algorithm is:

1) Tokenize the worst 3-word fragment, initialize a hash table and a correct word list
2) For each candidate word in the candidate word list:
   If the worst word is equal to the first token of the worst 3-word fragment then the first token is replaced with the candidate word then do the Google search on the modified worst fragment, lastly, stores the fragment and its number of returned results into the hash table
   For the second and the third token, they follow the same principle as the first token
3) Output the best fragment from the list of candidates if it has the most number of results.

E.g. “The hat sat” has 744 results returned, “The cat sat” has 69,300 results returned, “The car sat” has 28,800 results returned, the second fragment has the most number of returned results, therefore, it appear to be the most appropriate spelling suggestion.

4.2.6 Threshold issue

This initial work encounters the threshold problem that affects the performance of the work significantly. The threshold here is a fix upper bound for the number of returned results; it determines whether an n-word fragment is good or bad. For instance, the threshold is set to a hundred, if any n-word
fragment’s the number of returned search results is less than it, then the fragment is identified as bad fragment, otherwise it is good. However, there are many uncertainties exist when searching fragments on Google. For example, correct fragments may have less number of returned results than the threshold, or wrong fragment can have more number of returned results than the threshold. Consequently, it is hard to determine a fix threshold. Moreover, the threshold varies depends on the length of the input string. The longer the input string is, the harder to set the threshold. The reason is that long input has many n-word fragments.

4.2.7 Advantages and disadvantages of the initial work

Applying the combination of n-word fragments is able to detect bad words within an input phrase. However, this encounters the threshold problem, it is tough to set the threshold accurately to cover all the cases. This causes the program to produce false alarms; it would identify a correct word as a misspelling, or error as a correct word. And it only can detect single error in an input string.

4.3 Current Google spell checker

The current Google spell checker is implemented based on the initial work’s principles and is written in Java. It mainly focuses on detecting and automatically correcting multiple single-error misspellings (insertion, transposition, deletion and substitution) in an input string. Basically, it follows three steps: (1) detection of a fragment that contains misspelling(s), (2) generation of candidate fragments, and (3) ranking of candidate fragments. It makes the following assumptions: (1) a real-word error is unlikely to be semantically related to the whole text. (2) The user’s intended word should be semantically related to nearby words. Although it is not able to correct multiple-error misspellings (misspelling that contains more than one error types), it can identify them.

4.3.1 Three-word fragment technique

Unlike the initial work, the spell checker only applies the three-word fragment techniques. The reason for applying three-word fragment rather than 2 or 4-word fragment is, firstly, it has more contextual information than 2-word fragment. Secondly, although 4-word fragment is more accurate for detecting an error, it becomes less efficient when the later approaches applied. And it devises a three-word fragment algorithm to detect fragments that contain misspelling(s). Here is an initial step for applying three-word technique:

1) Set the input string to lower case
2) Initialize a hash map and tokenize the input string
3) Do Google search on each three-word fragment then store the fragment and its number of returned results in the map
4) Output the map

For the first step, the reason for setting the input string to lower case is that upper case would vary the number of returned results, for instance, “The chat sat” and “the chat sat”, the former one has more number of returned results than latter one. This would affect the program’s accuracy for detecting errors.

4.3.2 Error detection and correction method

This approach also uses a fix threshold to filter out fragments have more than 100,000 results. In this case, since it assumes that the input string is large or may contain significant number of errors, to set a slightly larger threshold than the one in the initial appears to be more appropriate. If too large the threshold is, the more fragments (including the good fragments) have to be handled. It firstly sorts the three-word fragments by comparing their number of returned results. Then it goes through each three-word fragment to remove the remaining fragments that also have the error. Consequently, it gains faster processing time. Once it has the wrong fragments, the next step is to detect the error in each wrong fragment. This approach applies the 2-word fragment technique to achieve the goal.

1. Divide each 3-word fragment into 2-word fragments
2. Search for each 2-word fragment on Google
3. Compare their number of returned results. The fragment has larger number of results is assigned as better fragment. And another one is assigned as worse fragment
4. Tokenize the 3-word fragment
5. Similar to the Error detection method
(Step2) in the initial work
6. Output the token has the most number of occurrence
e.g. "He can causes" is a wrong fragment, the 2-word fragment technique breaks it into “He can” and “can causes”. Then it retrieves these two fragment’s returned results, “He can” has 73,700,000 hits, and “can causes” has significant less hits, which is 55,700. Therefore, “can causes” and “He can” are seen as “bad” and “good” respectively. And then it counts the number of occurrences for each token in the wrong fragment to detect the error. In this case, the token “causes” has the most number of occurrences, so it is identified as error.

And then use the three-word fragment algorithm to do corrections:

1) For each fragment in the wrong fragments:
   Generate a list of candidate corrections for the error (see Step 2 from the initial work’s error correction method’s algorithm)

2) For each candidate fragment: do Google search on it and get the number of returned results

3) Output the best candidate fragment if it has the most number of returned results

E.g. “The chat sat on the mat.” is the input phrase, and “the chat sat.” is one of the wrong fragments in the list, the spell checker firstly generates all candidate corrections for the wrong token in the fragment, “the cat sat.” has 53,000 returned results, “the hat sat.” has 1,150 returned results, and the original one “the chat” however, the spell checker searches for significant number of n-word fragments during both the detection and correction phases. Especially in the correction phase, since a long text may contains significant number of errors, and an error may have many candidate corrections and the spell check needs to searches for each candidate correction on Google, it would beyond the query limitation very easily.

4.4 Comparisons between other existing experimental works and the Google spell checker
4.4.1 CRITIQUE/EPISTLE and the Google spell checker

By comparing the spell checker to the CRITIQUE/EPISTLE system, they use two distinct approaches to detect and correct real-word errors. One is context-based, while another one is relaxation and parser-based which detects and corrects grammatical anomalies.

In the case of CRITIQUE/EPISTLE system, it attempts to correct non-word errors before beginning it’s parsing phase. On the other hand, he Google spell checker detects and corrects non-word errors and real-word errors simultaneously. During the parsing procedure of CRITIQUE/EPISTLE system, it has to pass multiple tests. If the first parsing fails, then the system tries to relax some grammatical constraints, which might lead to a successful parsing. Finally, the parsing results are a tree that represents any relaxed constraints and suggested corrections. While, the spell checker attempts to use a threshold to category the fragments. CRITIQUE/EPISTLE employs an over 100,000-word online dictionary and about 300 complicated English grammar rules to parse each sentence in an input text. While, the spell checker applies 81520-word list and uses the web as a repository to generate a list of candidate corrections. CRITIQUE/EPISTLE system appears to be more accurate to detect and correct errors in texts that have a shorter average sentence length. On the other hand, the spell checker is more accurate for correcting errors in a longer average sentence length. Since the spell checker is context-dependent, short sentences have much less contextual information for it.

4.4.2 IBM developed system and the spell checker

The work resembles the Lancaster work is introduced by researchers at IBM uses similar principles as the spell checker does. It uses
probabilities of the co-occurrence of actual words to detect unlikely combinations of words. The principle of using probabilities derived from a very large corpus of text is similar to the number of returned search results technique applied to the spell checker. In the case of the probabilities, they determine what can occur after the first two words. In the case of the number of results, it determines which candidate correction can be the suggestion. Furthermore, the IBM system first generates all candidate corrections for a possible error, then calculates and compares the probability of each alternative to output the suggestion. The spell checker generates the number of returned results for each three-word fragment that contains the candidate correction, and then outputs the best fragment if it has the most returned results.

4.5 Evaluation

For this section, firstly, I am going to evaluate my Google spell checker’s performance for detecting and correcting real-word errors. In case of real-word error testing, the sentences contain semantic, syntactic errors, and simple typos, these are based on the four error types. The sentences are randomly derived from the essays about technology, science, people, history, politics and sports. The sentences have totally 1,179 words, and there are 220 misspellings within these words. The misspellings are mainly classified as noun-modifier disagreement, nonstandard verb form and simple typos. Each of them accounts for 39%, 25%, and 36% of the total number of errors respectively. And the testing results are classified as corrected and wrongly corrected errors, and the errors are detected but no spelling suggestions for them.

<table>
<thead>
<tr>
<th>Noun-modifier disagreement</th>
<th>Nonstandard verb form</th>
<th>Simple typos</th>
</tr>
</thead>
<tbody>
<tr>
<td>49%</td>
<td>60%</td>
<td>51%</td>
</tr>
</tbody>
</table>

Table 3. Google spell checker's ability for handling three types of real-word errors

In case of noun-modifier disagreement error, the spell checker is tested on if the subject is modified in number, for instance, singular or plural. “These book (books) are written by John.” In case of nonstandard verb form error, the verb has wrong form that is unlikely syntactic or semantic related to its nearby words. E.g. “He is going to presents (present) his views.” In case of simple typo, it is transformed by one of the error types into another valid word. But it is unlikely semantic related to its nearby words. E.g. “He an (and) me are going to play tennis.”

In table 3, it is obvious that the spell checker are more accurate for detecting and correcting nonstandard verb form errors, it is able to handle 60% of this type of error in the test set. On the other hand, it performs an average level for both noun-modifier disagreement and simple typos. In case of the former, it has approximate 49% error correction rate. In case of the latter, 51% of it are detected and corrected.

4.5.1 Example results

In this section, I give some examples of situations in which the spell checker succeeded and those in which it failed.

Example 1
The committee is now (not) prepared to grant your request.

Although the error produces a meaning other than that intended, it results in a perfectly well formed text. So the spell checker is not able to detect without knowledge or inference of the user’s intention.

Example 2
They later gathered volcanic ash from around the specimen and tested it to finds (find) the date. It turned out the fossils (fossil) was over 3 million year (years) old and there was about forty percents (percent) of the entire skeleton preserves (preserved).

In this example, the spell checker can handle 4 out of 5 errors. In theoretical, the errors “finds” and “year” in the wrong 3-word fragments “it to finds” and “3 million year” are unlikely syntactic to their nearby. In practical, in order to detect the errors, the fragments are further
divided into 2-word fragments, and then justify each 2-word fragment’s value. Google is sensitive to these types of grammatical errors. So for the wrong fragment “it to finds”, “it to” has significant larger number of hits than “to finds”, “finds” obviously has the wrong verb form.

Example 3
In both studies, disputes (despite) the many methodological problem (problems) identified by the researchers, both government (governments) were sufficiently persuade (persuaded) as to invests (invest) in national large-scale surveys.

In this case, the spell checker successfully identifies the wrong fragment, “many methodological problem”, but it wrongly corrects it. It thinks “many” is the error rather than “problem”. Because correct 2-word fragment may have less number of returned results than wrong fragment. Moreover, the drawback of 2-word fragment is that it contains limited contextual information. For the wrong fragment, “many methodological problems”, 3-word fragment technique can easily justify its value without knowledge of the user or the inference of the user’s intention. However, 2-word fragment technique has difficulties to do so. The fragment “many methodological” is detected by Google as an unlikely word combination. While, “methodological problem” is a perfectly correct fragment without “many”.

The error “government” in the fragment “both government were” is not caught by the spell checker due to the limitation of the 2-word fragment. The middle token will always 0 occurrence. But each token can have three different positions within the text. In other words, there are two fragments contain the error that occurs either the first or last position of the fragments, so there still has 66% of chance to detect the error.

Example 4
The characters (character) does not realize that he is will (ill). He has a sexual passion force (forces) children which he does note (not) realize they do not shared (share).

The spell checker detects the fragment “he is will” is an unlikely combination of words, even if it is syntactically correct without referring to the whole text. It corrects the wrong 3-word fragment, “note realize they”, because it has syntactic and semantic knowledge based on Google, it assumes that the combination of adverb and verb and pronoun has higher probability than any other kinds of combinations to occur in real context.

Example 5: Exception from Google
Since Google frequently throws the exception from service object: for input string, the spell checker may fail to work. In the GoogleSearch.wsdl file, Google uses name="estimatedTotalResultsCount" type="xsd:int"/> which is 32-bit signed integer, according to Data type Reference and expected to be less than 2,147,483,647. But recently Google increase number of pages counted. E.g. the word “message” has approximate 3,120,000,000 returned results, this exceeds the limit. The value should be changed to xsd:long. Unfortunately, Google no longer debug the Web API, so there have no solutions for this.

4.5.2 Non-word detection and correction ability

In table 4, I test the spell checker against Microsoft Word 2003, ispell and Jazzy based on business correspondences, student essays, and professional texts for detecting and correcting non-word errors. The errors are single errors that contain only one error type. The results are classified as corrected and wrongly corrected errors.

<table>
<thead>
<tr>
<th>Spell checking system</th>
<th>Corrected Errors</th>
<th>Wrongly corrected errors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Google spell checker</td>
<td>70%</td>
<td>13.2%</td>
</tr>
<tr>
<td>Word</td>
<td>96.6%</td>
<td>3.3%</td>
</tr>
<tr>
<td>ispell</td>
<td>86.3%</td>
<td>4.4%</td>
</tr>
<tr>
<td>Jazzy</td>
<td>91.6%</td>
<td>1.1%</td>
</tr>
</tbody>
</table>

Table 4. Google spell checker's ability for handling non-word errors

According to table 4, Google spell checker is less accurate than the other three systems. It has 70% successful correction rate. While other systems, word, ispell, and Jazzy account for 96.6%, 86.3%, and 91.6% correction rate respectively. In case of wrongly corrected errors, the spell checker miscorrect 13.2% of the errors, while other systems rarely miscorrect errors.

4.6 Future works and conclusions

Handling real-word errors is always a tough task for spell checking, the Google spell checker has been able to detect and correct real-word errors in an input string, which is a success. However, it is not able to handle the error in the middle of fragment, which is a shame. Number of hits and length of an input
vary the spell checker’s performance. And the spell checker corrects limited error types (insertion, deletion, transposition and reverse). Also, there has a time consuming issue, the longer the input string is, the slower response time the spell checker produces. In the future, I will make the spell checker more usable and accurate. I will devise a cleverer algorithm to handle the middle error, and add more error types to produce wider range of error detection and correction.

5. Acknowledgement

Thanks to my project supervisor Colin Johnson for the advice and comments during our weekly meetings.

6. References


