Real-Word Spelling Correction using Google Web 1T n-gram with Backoff

Aminul ISLAM
Department of Computer Science, SITE
University of Ottawa
Ottawa, ON, Canada
mdislam@site.uottawa.ca

Diana INKPEN
Department of Computer Science, SITE
University of Ottawa
Ottawa, ON, Canada
diana@site.uottawa.ca

Abstract:

We present a method for correcting real-word spelling errors using the Google Web 1T n-gram data set and a normalized and modified version of the Longest Common Subsequence (LCS) string matching algorithm. Our method is focused mainly on how to improve the correction recall (the fraction of errors corrected) while keeping the correction precision (the fraction of suggestions that are correct) as high as possible. Evaluation results on a standard data set show that our method performs very well.

Keywords:

Real-word; spelling correction; Google web 1T; n-gram

1. Introduction

Real-word spelling errors are words in a text that occur when a user mistakenly types a correctly spelled word when another was intended. Errors of this type may be caused by the writer's ignorance of the correct spelling of the intended word or by typing mistakes. Such errors generally go unnoticed by most spellcheckers as they deal with words in isolation, accepting them as correct if they are found in the dictionary, and flagging them as errors if they are not. This approach would be sufficient to correct the non-word error *myss* in "It doesn't know what the *myss* is all about." but not the real-word error *muss* in "It doesn't know what the *muss* is all about." To correct the latter, the spell-checker needs to make use of the surrounding context such as, in this case, to recognise that *fuss* is more likely to occur than *muss* in the context of *all about*. Ironically, errors of this type may even be caused by spelling checkers in the correction of non-word spelling errors when the *auto-correct* feature in some word-processing software sometimes silently change a non-word to the wrong real word [1], and sometimes when correcting a flagged error, the user accidentally make a wrong selection from the choices offered [2]. An extensive review of real-word spelling correction is given in [3, 1] and the problem of spelling correction more generally is reviewed in [4].

In this paper, we present a method for correcting real-word spelling error using the Google Web 1T n-gram data set [5][1], and a normalized and modified version of the Longest Common Subsequence (LCS) string matching algorithm (details are in section 3.1). Our intention is to focus on how to improve the correction recall while maintaining the correction precision as high as possible. The reason behind this intention is that if the recall for any method is around 0.5, this means that the method fails to correct around 50 percent of the errors. As a result, we can not completely rely on these type of methods, for that we need some type of human interventions or suggestions to correct the rest of the uncorrected errors. Thus, if we have a method that can correct almost 90 percent of the errors, even generating some extra candidates that are incorrect is more helpful to the human.

This paper is organized as follow: Section 2 presents a brief overview of the related work. Our proposed method is described in Section 3. Evaluation and experimental results are discussed in Section 4. We conclude in Section 5.

2. Related Work

Work on real-word spelling correction can roughly be classified into two basic categories: methods based on semantic information or human-made lexical resources, and methods based on machine learning or probability information. Our proposed method falls into the latter category.

2.1 Methods Based on Semantic Information

The 'semantic information' approach first proposed by [6] and later developed by [1] detected semantic anomalies, but was not restricted to checking words from predefined confusion sets. This approach was based on the observation that the words that a writer intends are generally semantically related to their surrounding words, whereas some types of real-word spelling errors are not.

2.2 Methods Based on Machine Learning

Machine learning methods are regarded as lexical disambiguation tasks and confusion sets are used to model the ambiguity between words. Normally, the machine learning and statistical approaches rely on pre-defined confusion sets, which are sets (usually pairs) of commonly confounded words, such as {*their*, *there*, *they're*} and {*principle*, *principal*}. [7], an example of a machine-learning method, combined the Winnow algorithm with weighted-majority voting, using nearby and adjacent words as features. Another example of a machine-learning method is that of [8].

[1]Details of the Google Web 1T data set can be found at www.ldc.upenn.edu/Catalog/docs/LDC2006T13/readme.txt.

2.3 Methods Based on Probability Information

[9] proposed a statistical method using word-trigram probabilities for detecting and correcting real-word errors without requiring predefined confusion sets. In this method, if the trigram-derived probability of an observed sentence is lower than that of any sentence obtained by replacing one of the words with a spelling variation, then we hypothesize that the original is an error and the variation is what the user intended.

[2] analyze the advantages and limitations of [9]'s method, and present a new evaluation of the algorithm, designed so that the results can be compared with those of other methods, and then construct and evaluate some variations of the algorithm that use fixed-length windows. They consider a variation of the method that optimizes over relatively short, fixed-length windows instead of over a whole sentence (except in the special case when the sentence is smaller than the window), while respecting sentence boundaries as natural breakpoints. To check the spelling of a span of d words requires a window of length d+4 to accommodate all the trigrams that overlap with the words in the span. The smallest possible window is therefore 5 words long, which uses 3 trigrams to optimize only its middle word. They assume that the sentence is bracketed by two BoS and two EoS markers (to accommodate trigrams involving the first two and last two words of the sentence). The window starts with its left-hand edge at the first BoS marker, and the [9]'s method is run on the words covered by the trigrams that it contains; the window then moves d words to the right and the process repeats until all the words in the sentence have been checked. As [9]'s algorithm is run separately in each window, potentially changing a word in each, [2]'s method as a side-effect also permits multiple corrections in a single sentence.

[10] proposed a trigram-based method for real-word errors without explicitly using probabilities or even localizing the possible error to a specific word. This method simply assumes that any word trigram in the text that is attested in the British National Corpus [11] is correct, and any unattested trigram is a likely error. When an unattested trigram is observed, the method then tries the spelling variations of all words in the trigram to find attested trigrams to present to the user as possible corrections. The evaluation of this method was carried out on only 7100 words of the Wall Street Journal corpus, with 31 errors introduced (i.e., one error in every approximately 200 words) obtaining a recall of 0.33 for correction, a precision of 0.05 and a F-measure of 0.086.

3. Proposed Method

The proposed method first tries to determine some probable candidates and then finds the best one among the candidates. We consider a string similarity function and a frequency value function in our method. The following sections present a detailed description of each of these functions, followed by the procedure to determine some probable candidates along with the procedure to find the best candidate.

3.1 String Similarity between Two Strings

We use the longest common subsequence (LCS) [12] measure with some normalization and small modifications for our string similarity measure. We use the same three different modified versions of LCS that [13] used, along with another modified version of LCS, and then take a weighted sum of these[2]. [14] showed that edit distance and the length of the longest common subsequence are special cases of n-gram distance and similarity, respectively. [15] normalized LCS by dividing the length of the longest common subsequence by the length of the longer string and called it longest common subsequence ratio (LCSR). But LCSR does not take into account the length of the shorter string which sometimes has a significant impact on the similarity score.

[13] normalized the *longest common subsequence* so that it takes into account the length of both the shorter and the longer string and called it *normalized longest common subsequence* (NLCS). We normalize NLCS in the following way as it gives better similarity value, as well as it is more computationally efficient:

$$v_1 = NLCS(s_i, s_j) = \frac{2 \times len(LCS(s_i, s_j))}{len(s_i) + len(s_j)} \quad (1)$$

While in classical LCS, the common subsequence needs not be consecutive, in spelling correction, a consecutive common subsequence is important for a high degree of matching. [13] used *maximal consecutive longest common subsequence* starting at character 1, $MCLCS_1$ and *maximal consecutive longest common subsequence* starting at any character n, $MCLCS_n$. $MCLCS_1$ takes two strings as input and returns the shorter string or maximal consecutive portions of the shorter string that consecutively match with the longer string, where matching must be from first character (character 1) for both strings. $MCLCS_n$ takes two strings as input and returns the shorter string or maximal consecutive portions of the shorter string that consecutively match with the longer string, where matching may start from any character (character n) for both of the strings. They normalized $MCLCS_1$ and $MCLCS_n$ and called it *normalized* $MCLCS_1$ ($NMCLCS_1$) and *normalized* $MCLCS_n$ ($NMCLCS_n$) respectively. Similarly, we normalize $NMCLCS_1$ and $NMCLCS_n$ in the following way:

$$v_2 = NMCLCS_1(s_i, s_j) = \frac{2 \times len(MCLCS_1(s_i, s_j))}{len(s_i) + len(s_j)} \quad (2)$$

$$v_3 = NMCLCS_n(s_i, s_j) = \frac{2 \times len(MCLCS_n(s_i, s_j))}{len(s_i) + len(s_j)} \quad (3)$$

[13] did not consider consecutive common subsequence ending at the last character, though $MCLCS_n$ sometimes covers this, but not always. We argue that the consecutive common subsequence ending at the last character is as significant as the consecutive common subsequence starting at the first character. So, we introduce the *maximal consecutive longest common subsequence* ending at the last character, $MCLCS_z$ (Algorithm 1). Algorithm 1, takes two strings as input and returns the shorter string or the maximal consecutive portions of the shorter string that consecutively matches with the longer string, where matching must end at the last character for both strings. We normalize $MCLCS_z$ and call it *normalized* $MCLCS_z$ ($NMCLCS_z$).

$$v_4 = NMCLCS_z(s_i, s_j) = \frac{2 \times len(MCLCS_z(s_i, s_j))}{len(s_i) + len(s_j)} \quad (4)$$

We take the weighted sum of these individual values v_1, v_2, v_3, and v_4 from equation (1), (2), (3) and (4), respectively,

[2] [13] use modified versions because in their experiments they obtained better results (precision and recall) for schema matching on a sample of data than when using the original LCS, or other string similarity measures.

to determine the string similarity score, where α_1, α_2, α_3, α_4 are weights and $\alpha_1 + \alpha_2 + \alpha_3 + \alpha_4 = 1$. Therefore, the similarity of the two strings, $S \in [0, 1]$ is:

$$S(s_i, s_j) = \alpha_1 v_1 + \alpha_2 v_2 + \alpha_3 v_3 + \alpha_4 v_4 \tag{5}$$

We heuristically set equal weights for most of our experiments[3]. Theoretically, $v_3 \geq v_2$ and $v_3 \geq v_4$.

Algorithm 1: $MCLCS_z$ (Maximal Consecutive LCS ending at the last character)

```
input  : s_i, s_j      /* s_i and s_j are input strings
         where |s_i| <= |s_j| */
output : str           /* str is the Maximal Consecutive
         LCS ending at the last character */
str <- NULL
c <- 1
while |s_i| >= c do
    x <- SubStr(s_i, -c, 1)    /* returns cth character
    of s_i from the end */
    y <- SubStr(s_j, -c, 1)    /* returns cth character
    of s_j from the end */
    if x = y then
        str <- SubStr(s_i, -c, c)
    else
        return str
    end
    increment c
end
```

3.2 Normalized Frequency Value

We determine the normalized frequency value of each candidate word for a single position with respect to all other candidates for the same position. If we find n replacements of a word w_i which are $\{w_{i1}, w_{i2}, \cdots, w_{ij}, \cdots, w_{in}\}$, and their frequencies $\{f_{i1}, f_{i2}, \cdots, f_{ij}, \cdots, f_{in}\}$, where f_{ij} is the frequency of a n-gram (where $n \in \{5, 4, 3, 2\}$ and any candidate word w_{ij} is a member of the n-gram), then we determine the normalized frequency value of any candidate word w_{ij} as the frequency of the n-gram containing w_{ij}, over the maximum frequency among all the candidate words for that position.

$$F(w_{ij}) = \frac{f_{ij}}{\max(f_{i1}, f_{i2}, \cdots, f_{ij}, \cdots, f_{in})} \tag{6}$$

3.3 Determining Candidate Words (Phase 1)

Our task is to correct real-word spelling error from an input text using Google Web 1T data set. First, we use Google 5-gram data set to find candidate words of the word having spelling error. If the 5-gram data set fails to generate at least one candidate word then we move forward to 4-gram data set or 3-gram data set or 2-gram data set if the preceding data set fails to generate at least one candidate word. Let us consider an input text W which after tokenization[4] has m words, i.e., $W = \{w_1, w_2, \ldots, w_i, \ldots, w_m\}$, where w_i $(i > 1)$[5] is the word having the spelling error. First, we discuss how we find the candidates for the word marked as a spelling error and, then we discuss the procedure to find the most relevant single candidate from several candidates.

3.3.1 Determining candidate words using the 5-gram data set

We use the following steps:

1. We define the term *cut off frequency* for word w_i as the frequency of the 5-gram w_{i-4} w_{i-3} w_{i-2} w_{i-1} w_i (where $m \geqslant i \geqslant 5$) in the Google Web 1T 5-grams, if the said 5-gram exists. Otherwise, we set the *cut off frequency* of w_i as 0. The intuition behind using the *cut off frequency* is the fact that, if the word is misspelled, then the correct one should have a higher frequency than the misspelled one in the context. Thus, using the *cut off frequency*, we isolate a large number of candidates that we do not need to process.

2. We find all the 5-grams (where only w_i is changed while w_{i-4}, w_{i-3}, w_{i-2} and w_{i-1} are unchanged), if any, having frequency greater than the *cut off frequency* of w_i (determined in step 1). Let us consider that we find n replacements of w_i which are $R_1 = \{w_{i1}, w_{i2}, \cdots, w_{in}\}$ and their frequencies $F_1 = \{f_{i1}, f_{i2}, \cdots, f_{in}\}$ where f_{ij} is the frequency of the 5-gram w_{i-4} w_{i-3} w_{i-2} w_{i-1} w_{ij}. If there is no such 5-gram (having frequency above the *cut off frequency*), our task is two-fold: we set *matched*←1, if there is at least one 5-gram, and we jump to step 5 or 6 or 7 that is yet to visit.

3. For each $w_{ij} \in R_1$, we calculate the string similarity between w_{ij} and w_i using equation (5) and then assign a weight using the following equation (7) only to the words that return the string similarity value greater than 0.5.

$$weight(w_i, w_{ij}) = \beta S(w_i, w_{ij}) + (1 - \beta) F(w_{ij}) \tag{7}$$

   Equation (7) is used to ensure a balanced weight between the string similarity function and the probability function, where β refers to how much importance we

[3]We use equal weights in several places in this paper in order to keep the system unsupervised. If development data would be available, we could adjust the weights.

[4]We need to tokenize the input sentence to make the n-grams formed using the tokens returned after the tokenization consistent with the Google n-grams. The input sentence is tokenized in a manner similar to the tokenization of the Wall Street Journal portion of the Penn Treebank. Notable exceptions include the following:

- Hyphenated word are usually separated, and hyphenated numbers usually form one token.
- Sequences of numbers separated by slashes (e.g., in dates) form one token.
- Sequences that look like urls or email addresses form one token.

[5]It means that we do not consider the first word as the word with spelling error. Though, our method could have worked for the first word too. We did not do it here due to efficiency reasons. Google n-grams are sorted based on the first word, then the second word, and so on. Based on this sorting, all Google 5-grams, 4-grams, 3-grams and 2-grams are stored in 117, 131, 97 and 31 different files, respectively. For example, all the 117 Google 5-gram files could have been needed to access a single word instead of accessing just one 5-gram file, that we do for any other words. This is because when the first word needs to be corrected, it might be in any file among those 117 5-gram files.

give to the string similarity function with respect to the frequency function[6].

4. We sort the words found in step 3 that were given weights, if any, in descending order by the assigned weights and keep only a fixed number[7] of words as the candidate words[8].

5. If this step is not visited yet then we follow step 1 to step 4 with the 5-gram $w_{i-3}\ w_{i-2}\ w_{i-1}\ w_i\ w_{i+1}$ if $m-1 \geqslant i \geqslant 4$. Otherwise, we go to next step.

6. If this step is not visited yet then we follow step 1 to step 4 with the 5-gram $w_{i-2}\ w_{i-1}\ w_i\ w_{i+1}\ w_{i+2}$ if $m-2 \geqslant i \geqslant 3$. Otherwise, we go to next step.

7. If this step is not visited yet then we follow step 1 to step 4 with the 5-gram $w_{i-1}\ w_i\ w_{i+1}\ w_{i+2}\ w_{i+3}$ if $m-3 \geqslant i \geqslant 2$. Otherwise, we go to next step.

8. If we find exactly one word in step 4, then return that word as the suggestion word and exit.

9. If we find more than one word in step 4, we go to section 3.5. Otherwise, if *matched*=1 then return *no suggestion* and exit.

3.3.2 Determining candidate words using the 4-gram data set

We use the following steps:

1. We define the term *cut off frequency* for word w_i as the frequency of the 4-gram $w_{i-3}\ w_{i-2}\ w_{i-1}\ w_i$ (where $m \geqslant i \geqslant 4$) in the Google Web 1T 4-grams, if the said 4-gram exists. Otherwise, we set the *cut off frequency* of w_i as 0.

2. We find all the 4-grams (where only w_i is changed while w_{i-3}, w_{i-2} and w_{i-1} are unchanged), if any, having frequency greater than the *cut off frequency* of w_i (determined in step 1). Let us consider that we find n replacements of w_i which are $R_1 = \{w_{i1}, w_{i2}, \cdots, w_{in}\}$ and their frequencies $F_1 = \{f_{i1}, f_{i2}, \cdots, f_{in}\}$ where f_{ij} is the frequency of the 4-gram $w_{i-3}\ w_{i-2}\ w_{i-1}\ w_{ij}$. If there is no such 4-gram, our task is two folds: we set *matched*$\leftarrow$1, if there is at least one 4-gram having any frequency and we jump to step 5 or 6 that is yet to visit.

3. For each $w_{ij} \in R_1$, we calculate the string similarity between w_{ij} and w_i using equation (5) and then assign a weight using equation (7) only to the words that return the string similarity value greater than 0.5.

4. We sort the words found in step 3 that were given weights, if any, in descending order by the assigned weights and keep only a fixed number of words as the candidate words.

5. If this step is not visited yet then we follow step 1 to step 4 with the 4-gram $w_{i-2}\ w_{i-1}\ w_i\ w_{i+1}$ if $m-1 \geqslant i \geqslant 3$. Otherwise, we go to next step.

6. If this step is not visited yet then we follow step 1 to step 4 with the 4-gram $w_{i-1}\ w_i\ w_{i+1}\ w_{i+2}$ if $m-2 \geqslant i \geqslant 2$. Otherwise, we go to next step.

7. If we find exactly one word in step 4, then return that word as the suggestion word and exit.

8. If we find more than one word in step 4, we go to section 3.5. Otherwise, if *matched*=1 then return *no suggestion* and exit.

3.3.3 Determining candidate words using the 3-gram data set

We use the following steps:

1. We define the term *cut off frequency* for word w_i as the frequency of the 3-gram $w_{i-2}\ w_{i-1}\ w_i$ (where $m \geqslant i \geqslant 3$) in the Google Web 1T 3-grams, if the said 3-gram exists. Otherwise, we set the *cut off frequency* of w_i as 0.

2. We find all the 3-grams (where only w_i is changed while w_{i-2} and w_{i-1} are unchanged), if any, having frequency greater than the *cut off frequency* of w_i (determined in step 1). Let us consider that we find n replacements of w_i which are $R_1 = \{w_{i1}, w_{i2}, \cdots, w_{in}\}$ and their frequencies $F_1 = \{f_{i1}, f_{i2}, \cdots, f_{in}\}$ where f_{ij} is the frequency of the 3-gram $w_{i-2}\ w_{i-1}\ w_{ij}$. If there is no such 3-gram, our task is two folds: we set *matched*$\leftarrow$1, if there is at least one 3-gram having any frequency and we jump to step 5, if it is yet to visit.

3. For each $w_{ij} \in R_1$, we calculate the string similarity between w_{ij} and w_i using equation (5) and then assign a weight using equation (7) only to the words that return the string similarity value greater than 0.5.

4. We sort the words found in step 3 that were given weights, if any, in descending order by the assigned weights and keep only a fixed number of words as the candidate words.

5. If this step is not visited yet then we follow step 1 to step 4 with the 3-gram $w_{i-1}\ w_i\ w_{i+1}$ if $m-1 \geqslant i \geqslant 2$. Otherwise, we go to next step.

6. If we find exactly one word in step 4, then return that word as the suggestion word and exit.

7. If we find more than one word in step 4, we go to section 3.5. Otherwise, if *matched*=1 then return *no suggestion* and exit.

3.3.4 Determining candidate words using the 2-gram data set

We use the following steps:

[6] We give more importance to string similarity function with respect to frequency value function throughout the section of 'determining candidate words' to have more candidate words so that the chance of including the target word into the set of candidate words gets higher. For this reason, we heuristically set β=0.85 in equation (7) instead of setting β=0.5.

[7] For our experiment, we heuristically set this number as 10. If we lower this number (say 2 or 3) then there is a chance that sometimes the most appropriate candidate (solution word) fails to be included in the candidate list.

[8] Sometimes the top candidate word might be either a plural form or a past participle form of the original word. Or even it might be a high frequency function word (e.g., *the*). We omit these type of words from the candidacy.

1. We define the term *cut off frequency* for word w_i as the frequency of the 2-gram $w_{i-1}\ w_i$ (where $m \geqslant i \geqslant 2$) in the Google Web 1T 2-grams, if the said 2-gram exists. Otherwise, we set the *cut off frequency* of w_i as 0.

2. We find all the 2-grams (where only w_i is changed while w_{i-1} is unchanged) having frequency greater than the *cut off frequency* of w_i (determined in step 1). Let us consider that we find n replacements of w_i which are $R_1 = \{w_{i1}, w_{i2}, \cdots, w_{in}\}$ and their frequencies $F_1 = \{f_{i1}, f_{i2}, \cdots, f_{in}\}$ where f_{ij} is the frequency of the 2-gram $w_{i-1}\ w_{ij}$. If there is no such 2-gram, we set *matched*←1, if there is at least one 2-gram having any frequency.

3. For each $w_{ij} \in R_1$, we calculate the string similarity between w_{ij} and w_i using equation (5) and then assign a weight using equation (7) only to the words that return the string similarity value greater than 0.5.

4. We sort the words found in step 3 that were given weights, if any, in descending order by the assigned weights and keep only a fixed number of words as the candidate words.

5. If we find exactly one word in step 4, then return that word as the suggestion word and exit.

6. If we find more than one word in step 4, we go to section 3.5. Otherwise, if *matched*=1 then return *no suggestion* and exit. Otherwise, we proceed to phase 2 (section 3.4).

3.4 Determining Candidate Words (Phase 2)

The question of why we use phase 2 is best understood by the example "$\cdots$ by releasing the WPPSS *retort*." (Table 1) where *retort* is the observed word that needs to be corrected. But, there is no such 5-gram where *retort* can be changed by following the condition of having the string similarity between *retort* and w_i be at least 0.5 and keeping "by releasing the WPPSS" unchanged. Similarly, there is no such 4-gram, 3-gram and 2-gram where *retort* can be changed by following the same previous condition and keeping "releasing the WPPSS", "the WPPSS" and "WPPSS" unchanged respectively. The reason of the unavailability of such n-grams is that "WPPSS" is not a very common word in the Google Web 1T data set.

To solve this issue is straightforward. We follow phase 1 with some small changes: instead of trying to find all the n-grams ($n \in \{5, 4, 3, 2\}$) where only w_i is changed while keeping all of $\{\cdots, w_{i-2}, w_{i-1}\}$ unchanged, we try to find all the n-grams ($n \in \{5, 4, 3, 2\}$) where w_i, as well as any but the first member of $\{\cdots, w_{i-2}, w_{i-1}\}$ are changed while keeping the rest of $\{\cdots, w_{i-2}, w_{i-1}\}$ unchanged.

3.5 Determining the Suggestion Word

We use this section only if we have more than one candidate word found in section 3.3 or section 3.4. Let us consider that we find n candidate words of w_i in section 3.3 or section 3.4 which are $\{w_{i1}, w_{i2}, \cdots, w_{ij}, \cdots, w_{in}\}$. For each w_{ij}, we use the string similarity value between w_{ij} and w_i (already calculated using equation (5)) and the normalized frequency value of w_{ij} (already calculated using equation (6)) and then calculate the weight value using equation (7) by setting $\beta = 0.5$. We find the word having the maximum weight value as the target suggestion word which is:

$$Suggestion\ Word = \underset{w_{ij}}{\operatorname{argmax}}\ weight(w_i, w_{ij}) \tag{8}$$

4. Evaluation and Experimental Results

We used as test data the same data that [2] used in their evaluation of [9] method, which in turn was a replication of the data used by [6] and [1] to evaluate their methods.

The data consisted of 500 articles (approximately 300,000 words) from the 1987−89 *Wall Street Journal* corpus, with all headings, identifiers, and so on removed; that is, just a long stream of text. It is assumed that this data contains no errors; that is, the *Wall Street Journal* contains no malapropisms or other typos. In fact, a few typos (both non-word and real-word) were noticed during the evaluation, but they were small in number compared to the size of the text.

Malapropisms were randomly induced into this text at a frequency of approximately one word in 200. Specifically, any word whose base form was listed as a noun in WordNet (but regardless of whether it was used as a noun in the text; there was no syntactic analysis) was potentially replaced by any spelling variation found in the lexicon of the *ispell* spelling checker[9]. A *spelling variation* was defined as any word with an *edit distance* of 1 from the original word; that is, any single-character insertion, deletion, or substitution, or the transposition of two characters, that results in another real word. Thus, none of the induced malapropisms were derived from closed-class words, and none were formed by the insertion or deletion of an apostrophe or by splitting a word. Though [2] mentioned that the data contained 1402 inserted malapropisms, there were only 1391 malapropisms.

Because it had earlier been used for evaluating [9]'s tri-gram method, which operates at the sentence level, the data set had been divided into three parts, without regard for article boundaries or text coherence: sentences into which no malapropism had been induced; the original versions of the sentences that received malapropisms; and the malapropized sentences. In addition, all instances of numbers of various kinds had been replaced by tags such as <INTEGER>, <DOLLAR VALUE>, and <PERCENTAGE VALUE>. Actual (random) numbers or values were restored for these tags. Some spacing anomalies around punctuation marks were corrected. A detailed description of this data can be found in [16, 2].

Some examples of successful and unsuccessful corrections, using Google 5-grams, are shown in Table 1. Some of the malapropisms created were "unfair" in the sense that no automatic procedure could reasonably be expected to see the error. The canonical case is the substitution of *million* for *billion*, or vice versa[10]; another is *employee* for *employer*, or vice versa, in many (but not all) contexts. In some cases, the substitution was merely a legitimate spelling variation of the same word (e.g., *labour* for *labor*).

Table 2 shows some examples of successful and unsuccessful corrections using Google 4-grams where Google 5-grams fail to generate any suggestion. In Table 3 and Table 4, some

[9] Ispell is a fast screen-oriented spelling checker that shows you your errors in the context of the original file, and suggests possible corrections when it can figure them out.

[10] One such example is shown in FALSE NEGATIVE section in Table 1. There are 40 malapropisms in the data set related with *million*/*billion*.

SUCCESSFUL CORRECTION:
··· chance to mend his *fencers* → fences [fences] with Mr. Jefferies ···
··· employees is the largest *employee* → employer [employer] in Europe and ···
SUCCESSFUL CORRECTION (in Second Phase):
··· by releasing the WPPSS *retort* → report [report].
··· tests comparing its potpourri *covert* → cover [cover] with the traditional ···
FALSE POSITIVE CORRECTION:
··· can almost see the *firm* → fire [farm] issue receding.
··· the Senate to support *aim* → him [aid] for the Contras ···
FALSE NEGATIVE:
I trust that the *contract* [contrast] between the American ···
··· as much as <DOLLAR_VALUE> *billion* [million].

Table 1: Examples of successful and unsuccessful corrections using Google 5-grams. Italics indicate the observed word, arrow indicates the correction, square brackets indicate the intended word.

SUCCESSFUL CORRECTION:
··· one of a corporate *raiser* → raider [raider]'s five most ···
··· of rumors about insider *tracing* → trading [trading] in Pillsbury options ···
SUCCESSFUL CORRECTION (in Second Phase):
··· , for the Belzberg *brothels* → brothers [brothers]' First City ···
··· typical of Iowa's *food* → mood [mood] a month before ···
FALSE POSITIVE CORRECTION:
··· I'm uncomfortable *tacking* → talking [taking] a lot of ···
··· to approve a formal *teat* → test [text] of their proposed ···
FALSE NEGATIVE:
A lot of the *optimisms* [optimists] were washed out ···
··· published its support of *patent* [patient]-funded research ···

Table 2: Examples of successful and unsuccessful corrections using Google 4-grams. For these examples, Google 5-grams fail to generate any suggestion.

examples of successful and unsuccessful corrections using Google 3-grams (where Google 5-grams and 4-grams fail to generate any suggestion) and using Google 2-grams (where Google 5-grams, 4-grams and 3-grams fail to generate any suggestion) are shown respectively.

For each error, our method returns either a suggestion (which is either correct[11] or wrong[12]) or *no suggestion*[13].

[11]A returned suggestion which is correct is also known as *true positive*.

[12]A returned suggestion which is wrong is also known as *false positive*.

[13]Also known as *false negative*. Generating *no suggestion* means that the method does not find any better suggestion than the observed word. In other words, the method thinks that the observed word is indeed a correct one.

SUCCESSFUL CORRECTION:
··· Disappointment turned to dire *traits* → straits [straits] when Vestron's ···
··· pay <DOLLAR_VALUE> million in *destitution* → restitution [restitution] related to contract ···
SUCCESSFUL CORRECTION (in Second Phase):
··· its Ally & Gargano *unity* → unit [unit] settled their suit ···
··· 23 5/8 after rising 23 3/8 *prints* → points [points] Tuesday.
FALSE POSITIVE CORRECTION:
··· working on improving his *lent* → talent [left].
··· company feels the 23 *mate* → rate [date] is for the ···
FALSE NEGATIVE:
··· town saloon after the *battle* [cattle] roundup.
··· be this rock-*firm* [film] orgy, an ···

Table 3: Examples of successful and unsuccessful corrections using Google 3-grams. For these examples, Google 5-grams and 4-grams fail to generate any suggestion.

SUCCESSFUL CORRECTION:
··· raider or Zurn management *making* → taking [taking] shares out of ···
··· Silesia, said Solidarity *advisor* → adviser [adviser] Jacek Kuron.
SUCCESSFUL CORRECTION (in Second Phase):
Mr. Mutert *votes* → notes [notes] that investors have ···
FALSE POSITIVE CORRECTION:
··· relieved if Super Tuesday *toughs* → tours [coughs] up a front ···
··· be reserved about indiscriminate *clays* → ways [plays] in some groups ···
FALSE NEGATIVE:
··· aimed at clearing out *overstuffing* [overstaffing] left by previous ···
··· NMS rose (8.4 *billion* [million]) than fell ···

Table 4: Examples of successful and unsuccessful corrections using Google 2-grams. For these examples, Google 5-grams, 4-grams and 3-grams fail to generate any suggestion.

Figure 1 shows the number of errors where either a suggestion or *no suggestion* is generated for different combinations of n-grams used. To give an example, using only 5-grams, each of 505 errors either generate a suggestion or *no suggestion*. It also means that in the next 5-4-gram combination, we only process 886 errors[14] (i.e., 1391-505). Figure 1 validates the intuition behind using a combination of n-grams rather using only n-grams (e.g., 5-grams) by showing that while single 5-gram generates either a suggestion or *no suggestion* for only 505 errors, a combination of 5-4-3-2-$5'$-$4'$-

[14]We use the result of the previous 5-grams in 5-4-gram combination, thus only use 4-grams in this combination.

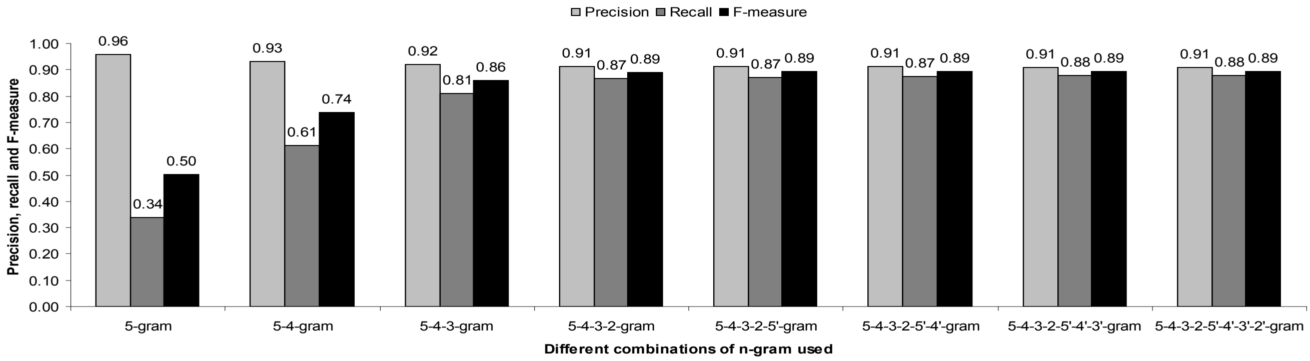

Figure 3: Precision, recall and F-measure for different combinations of n-grams used.

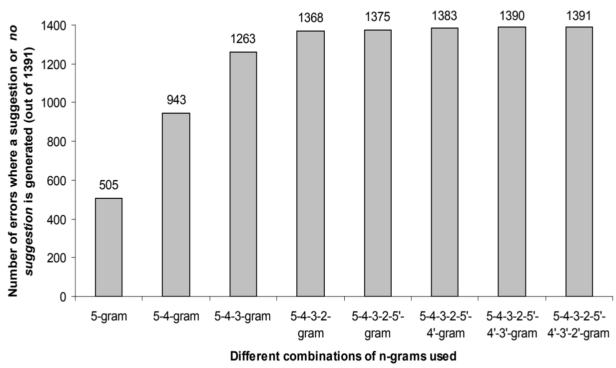

Figure 1: Number of errors where a suggestion or *no suggestion* is generated for different combinations of n-grams used. Apostrophe ($'$) is used to denote the n-grams used in phase 2. x-y-$\cdots$ z-gram means that we use x-grams, y-grams, $\cdots$ and z-grams.

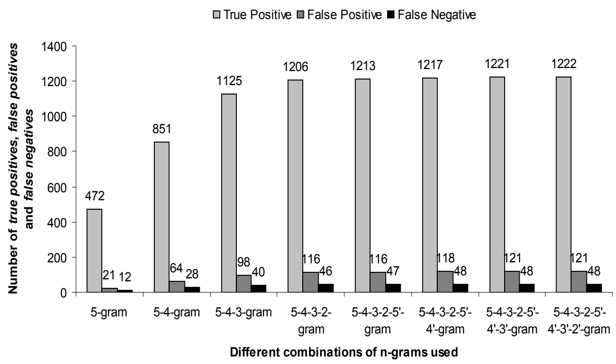

Figure 2: Number of true positives, false positives and false negatives for different combinations of n-grams used.

$3'$-$2'$-gram generates the same for all 1391 errors.

Figure 2 breaks down the numbers shown in Figure 1 into *true positive*, *false positive* and *false negative*. For example, using only 5-grams, we get 493 suggestions, 472 out of them are correct and 21 are incorrect, along with 12 *no suggestion*. Figure 2 also shows that a combination of 5-4-3-2-$5'$-$4'$-$3'$-$2'$-gram generates 1343 suggestions, 1222 out of them are correct and 121 are incorrect, along with 48 *no suggestion*.

The performance is measured using *Precision* (P), *Recall* (R) and *F-measure* (F):

$$P = \frac{\textit{number of correct suggestions returned}}{\textit{number of suggestions returned}}$$

$$R = \frac{\textit{number of correct suggestions returned}}{\textit{total number of errors in the collection}}$$

$$F = \frac{2PR}{P+R}$$

Figure 3 shows *precision*, *recall* and *F-measure* for different combinations of n-gram used. We get highest *precision* (0.96) when using only 5-gram which is obvious because 5-gram uses maximum possible context words (which is four) and as a result the chance of getting highest ratio between the number of correct suggestions returned and the number of suggestions returned increases. But the *recall* at this level is very poor (only 0.34). Figure 3 demonstrates how *recall* gets better using different combinations of n-gram while keeping *precision* as high as possible. Using 5-gram to a combination of 5-4-3-2-gram, we get a significant improvement of *recall* but after that (i.e., a combination of 5-4-3-2-gram to a combination of 5-4-3-2-$5'$-$4'$-$3'$-$2'$-gram), we get only 0.01 *recall* increase.

We cannot directly compare our results with the correction results from previous work, because in that work the correction was run on the results of the detection module, cumulating the errors, while our correction module ran on the correctly-flagged spelling errors. Still, we indirectly try to compare our results with the previous work. Table 5 shows our method's results on the described data set compared with the results for the trigram method of [2] and the lexical cohesion method of [1]. The data shown here for trigram method are not from [2], but rather are later results following some corrections reported in [16][15]. That

[15]The result (detection *recall*=0.544, detection *precision*=0.528, correction *recall*=0.491, correction *precision*=0.503) mentioned in [16] seems to have some inconsistency in correction *recall* and correction *precision*. Detection true positives (762) and detection false positives (681) can be calculated from detection *precision* and *recall*. Correction true positives and correction false positives are 688 and 755, respectively, given that the correction *recall* is 0.491. Thus, correction *precision* is 0.477.

Detection			correction		
R	P	F	R	P	F
Lexical cohesion[1]					
0.306	0.225	0.260	0.281	0.207	0.238
Trigrams[2]					
0.544	0.528	0.536	0.491	0.477	0.484
Google n-grams					
-	-	-	0.88	0.91	0.89

Table 5: A comparison of recall, precision, and F-measure for three methods of malapropism detection and correction on the same data set.

the corrected result of [2] can detect 762 errors and thus correct 688 errors out of these 762 detected errors means each of the correction *precision*, *recall* and F-measure is 0.9. It is obvious that the performance of correcting the rest of the undetected errors will not be the same as correcting the detected errors because these errors are difficult to correct since they are difficult to detect in the first place. Still, the correction performance of our proposed method is comparable to the correction performance of the method that runs on the results of the detection module, cumulating the errors.

Moreover, considering the fact[16] that 85 malapropisms out of 1391 created were "unfair" in the sense that no automatic procedure could reasonably be expected to see the error[16], to have a method generating 1222 correct suggestions along with 121 wrong suggestions and 48 *no suggestion* could be useful.

5. Conclusions

Our purpose in this paper was the development of a high-quality correction module. The Google n-grams proved to be very useful in correcting real-word errors. When we tried with only 5-grams the precision (0.96) was good, though the recall (0.34) was too low. Having sacrificed a bit of the precision score, our proposed combination of n-grams method achieves a very good recall (0.88) while maintaining the precision at 0.91. Our attempts to improve the correction recall while maintaining the precision as high as possible are helpful to the human correctors who post-edit the output of the real-word spell checker. If there is no postediting, at least more errors get corrected automatically. Our method could also correct misspelled words, not only malapropism, without any modification. In future work, we plan to add a detection module and extend our method to allow for deleted or inserted words, and to find the corrected strings in the Google Web 1T n-grams. In this way we will be able to correct grammar errors too.

Acknowledgments

This work is funded by the Natural Sciences and Engineering Research Council of Canada. We want to thank Professor Graeme Hirst from the Department of Computer Science, University of Toronto, for providing the evaluation data set.

References

[16] Though we did not consider this fact in the evaluation procedure.

[1] G. Hirst and A. Budanitsky, "Correcting real-word spelling errors by restoring lexical cohesion," *Natural Language Engineering*, vol. 11, pp. 87–111, March 2005.

[2] L. A. Wilcox-O'Hearn, G. Hirst, and A. Budanitsky, "Real-word spelling correction with trigrams: A reconsideration of the mays, damerau, and mercer model," in *Proceedings, 9th International Conference on Intelligent Text Processing and Computational Linguistics (CICLing-2008) (Lecture Notes in Computer Science 4919, Springer-Verlag)* (A. Gelbukh, ed.), (Haifa), pp. 605–616, February 2008.

[3] J. Pedler, *Computer Correction of Real-word Spelling Errors in Dyslexic Text*. PhD thesis, Birkbeck, London University, 2007.

[4] K. Kukich, "Technique for automatically correcting words in text," *ACM Comput. Surv.*, vol. 24, no. 4, pp. 377–439, 1992.

[5] T. Brants and A. Franz, "Web 1T 5-gram corpus version 1.1.," tech. rep., Google Research, 2006.

[6] G. Hirst and D. St-Onge, *WordNet: An electronic lexical database*, ch. Lexical chains as representations of context for the detection and correction of malapropisms, pp. 305–332. Cambridge, MA: The MIT Press, 1998.

[7] A. R. Golding and D. Roth, "A winnow-based approach to context-sensitive spelling correction," *Machine Learning*, vol. 34, no. 1-3, pp. 107–130, 1999.

[8] A. J. Carlson, J. Rosen, and D. Roth, "Scaling up context-sensitive text correction," in *Proceedings of the Thirteenth Conference on Innovative Applications of Artificial Intelligence Conference*, pp. 45–50, AAAI Press, 2001.

[9] E. Mays, F. J. Damerau, and R. L. Mercer, "Context based spelling correction," *Information Processing and Management*, vol. 27, no. 5, pp. 517–522, 1991.

[10] S. Verberne, "Context-sensitive spell checking based on word trigram probabilities," Master's thesis, University of Nijmegen, February-August 2002.

[11] L. Burnard, *Reference Guide for the British National Corpus (World Edition)*, October 2000. www.natcorp.ox.ac.uk/docs/userManual/urg.pdf.

[12] L. Allison and T. Dix, "A bit-string longest-common-subsequence algorithm," *Information Processing Letters*, vol. 23, pp. 305–310, 1986.

[13] A. Islam and D. Inkpen, "Semantic text similarity using corpus-based word similarity and string similarity," *ACM Transactions on Knowledge Discovery from Data*, vol. 2, no. 2, pp. 1–25, 2008.

[14] G. Kondrak, "N-gram similarity and distance," in *Proceedings of the 12h International Conference on String Processing and Information Retrieval*, (Buenos Aires, Argentina), pp. 115–126, 2005.

[15] I. D. Melamed, "Bitext maps and alignment via pattern recognition," *Computational Linguistics*, vol. 25, no. 1, pp. 107–130, 1999.

[16] G. Hirst, "An evaluation of the contextual spelling checker of microsoft office word 2007," January 2008. http://ftp.cs.toronto.edu/pub/gh/Hirst-2008-Word.pdf.