Real-Word Spelling Correction using Google Web 1T n-gram Data Set

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.

Categories and Subject Descriptors

I.2.7 [**Artificial Intelligence**]: Natural Language Processing—*Text analysis*; I.2.1 [**Artificial Intelligence**]: Applications and Expert Systems—*Natural language interfaces*

General Terms

Algorithms, Experimentation, Performance

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. 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 [4], and sometimes when correcting a flagged error, the user accidentally make a wrong selection from the choices offered [10]. An extensive review of real-word spelling correction is given in [4].

CIKM'09, November 2–6, 2009, Hong Kong, China.

In this paper, we present a method for correcting real-word spelling error using the Google Web 1T n-gram data set [1][1], and a normalized and modified version of the Longest Common Subsequence (LCS) string matching algorithm. We do not try to correct an error, if any, in the first word as it is not computationally feasible to search in the Google Web 1T n-grams while keeping the first word in the n-gram as a variable. Our intention is to focus on how to improve the correction recall while maintaining the correction precision as high as possible.

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.

The 'semantic information' approach first proposed by [5] and later developed by [4] 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.

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*}. Golding and Roth [2], an example of a machine-learning method, combined the Winnow algorithm with weighted-majority voting, using nearby and adjacent words as features.

Mays et al. [8] proposed a statistical method using word-trigram probabilities for detecting and correcting real-word errors without requiring predefined confusion sets. Wilcox-O'Hearn et al. [10] analyze the advantages and limitations of [8]'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. Ver-

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

berne [9] 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 is correct, and any unattested trigram is a likely error.

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 same string similarity measure that [6, 7] used with the following different normalization as it gives better similarity value, as well as it is more computationally efficient:

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

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

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

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

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, 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:

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

We heuristically set equal weights for most of our experiments[2]. Theoretically, $v_3 \geq v_2$ and $v_3 \geq v_4$.

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.

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

3.3 Determining Candidate Words (Phase 1)

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 has m words, i.e., $W = \{w_1, w_2, \ldots, w_i, \ldots, w_m\}$, where w_i $(i > 1)$[3] is the word having the spelling error.

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*$\leftarrow$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.
$$weight(w_i, w_{ij}) = \beta S(w_i, w_{ij}) + (1 - \beta) F(w_{ij}) \quad (7)$$
Equation (7) is used to ensure a balanced weight between the string similarity function and the frequency value function, where β refers to how much importance we give to the string similarity function with respect to the frequency value function.
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 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.

[2]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.

[3]It means that we do not consider the first word as the word with spelling error.

3.3.2 Determining candidate words using the 4-gram, 3-gram and 2-gram data set

We use the same steps described in section 3.3.1 to determine the candidate words using the 4-gram, 3-gram and 2-gram data set. If we find more than one candidate word, we go to section 3.5. Otherwise, if *matched*=1 then return *no suggestion* and exit. Otherwise, we proceed to phase 2.

3.4 Determining Candidate Words (Phase 2)

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, and 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:

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

4. EVALUATION AND EXPERIMENTAL RESULTS

We used as test data the same data that [10] used in their evaluation of [8] method, which in turn was a replication of the data used by [5] and [4] 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.

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 was potentially replaced by any spelling variation found in the lexicon of the *ispell* spelling checker[4]. A *spelling variation* was defined as any word with an *edit distance* of 1 from the original word. Though [10] mentioned that the data contained 1402 inserted malapropisms, there were only 1391 malapropisms. A detailed description of this data can be found in [3, 10].

Some examples of successful and unsuccessful corrections, using Google 5-grams, are shown in Table 1. For each error, our method returns either a suggestion (which is either correct[5] or wrong[6]) or *no suggestion*[7]. Figure 1 shows the number of errors where either a suggestion or *no suggestion* is generated for different combinations of n-grams used. Figure 2 breaks down the numbers shown in Figure 1 into *true positive*, *false positive* and *false negative*.

[4]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.
[5]A returned correct suggestion is also known as *true positive*.
[6]A returned wrong suggestion is also known as *false positive*.
[7]Also known as *false negative*.

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.

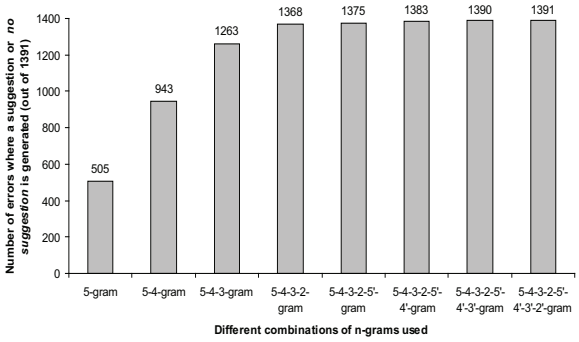

Figure 1: Number of errors where a suggestion or *no suggestion* is generated for different combinations of n-grams used. Apostrophe ($'$) is used to denote the n-grams used in phase 2. x-y-$\cdots$ z-gram means that we use x-grams, y-grams, $\cdots$ and z-grams.

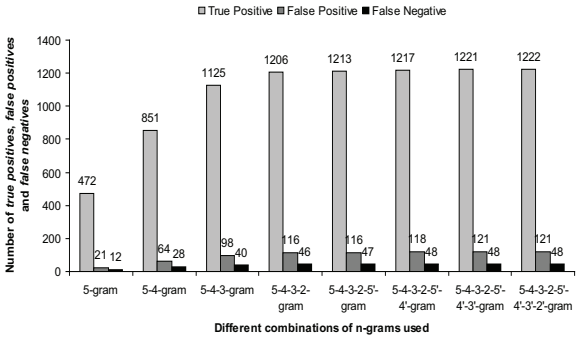

Figure 2: Number of true positives, false positives and false negatives for different combinations of n-grams used.

The performance is measured using *Precision* (P), *Recall* (R) and *F-measure* (F). The fraction of suggestions returned that are correct is the correction *precision* and the fraction of errors corrected is the correction *recall*. Figure 3 shows *precision*, *recall* and *F-measure* for different combinations of n-gram used. Figure 3 demonstrates how *recall* gets

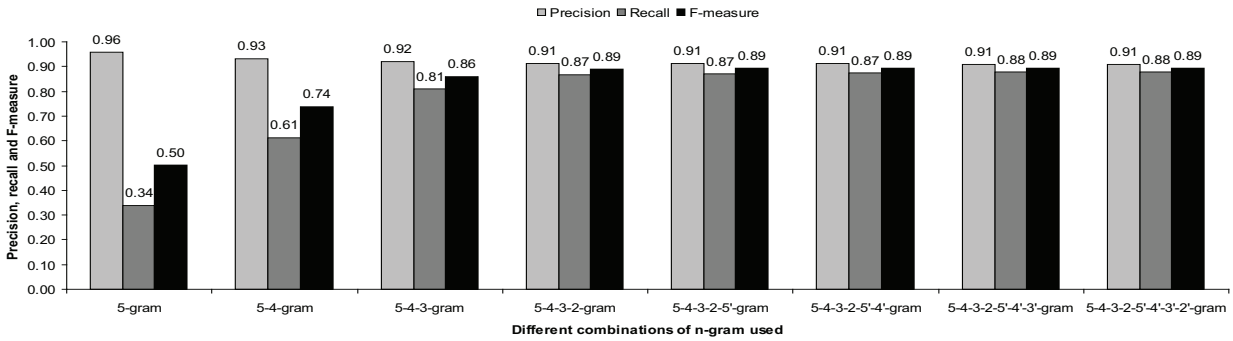

Figure 3: Precision, recall and F-measure for different combinations of n-grams used.

Detection			correction		
R	P	F	R	P	F
Lexical cohesion [4]					
0.306	0.225	0.260	0.281	0.207	0.238
Trigrams [10]					
0.544	0.528	0.536	0.491	0.477	0.484
Google *n*-grams					
-	-	-	0.88	0.91	0.89

Table 2: A comparison of recall, precision, and F-measure for three methods of malapropism detection and correction on the same data set.

better using different combinations of n-gram while keeping *precision* as high as possible.

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 2 shows our method's results on the described data set compared with the results for the trigram method of [10] and the lexical cohesion method of [4]. The data shown here for trigram method are not from [10], but rather are later results following some corrections reported in [3]. That the corrected result of [10] 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.

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.

6. REFERENCES

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

[2] A. R. Golding and D. Roth. A winnow-based approach to context-sensitive spelling correction. *Machine Learning*, 34(1-3):107–130, 1999.

[3] 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.

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

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

[6] A. Islam and D. Inkpen. Semantic text similarity using corpus-based word similarity and string similarity. *ACM Transactions on Knowledge Discovery from Data*, 2(2):1–25, 2008.

[7] A. Islam and D. Inkpen. Real-word spelling correction using Google Web 1T 3-grams. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, pages 1241–1249, Singapore, August 2009.

[8] E. Mays, F. J. Damerau, and R. L. Mercer. Context based spelling correction. *Information Processing and Management*, 27(5):517–522, 1991.

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

[10] 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 A. Gelbukh, editor, In *Proceedings of CICLing-2008 (LNCS 4919, Springer-Verlag)*, pages 605–616, Haifa, February 2008.