Correcting Different Types of Errors in Texts

Aminul Islam and Diana Inkpen

University of Ottawa, Ottawa, Canada
{mdislam,diana}@site.uottawa.ca

Abstract. This paper proposes an unsupervised approach that automatically detects and corrects a text containing multiple errors of both syntactic and semantic nature. The number of errors that can be corrected is equal to the number of correct words in the text. Error types include, but are not limited to: spelling errors, real-word spelling errors, typographical errors, unwanted words, missing words, prepositional errors, punctuation errors, and many of the grammatical errors (e.g., errors in agreement and verb formation).

Key words: Text Error Correction, Detection, Unsupervised, Google Web 1T 5-grams

1 Introduction

Most approaches to text correction are for only one or at best for a few types of errors. To the best of our knowledge, there is no fully-unsupervised approach that corrects a text having multiple errors of both syntactic and semantic nature. Syntactic errors refer to all kinds of grammatical errors. For example, in the sentence, "*Our method correct real-word spelling errors.*", there is an error of syntactic nature in subject-verb agreement, whereas, in the sentence, "*She had a cup of powerful tea.*", the word '*strong*' is more appropriate than the word '*powerful*' in order to convey the proper intended meaning of the sentence, based on the context. The latter is an example of a semantic error.

In this paper, a more general unsupervised statistical method for automatic text error detection and correction, done in the same time, using the Google Web 1T 5-gram data set [1] is presented. The proposed approach uses the three basic text correction operations: *insert*, *delete*, and *replace*. We use the following three strict assumptions for the input text that needs to be corrected: (1) The first token is a word[1]. (2) There should be at least three words in an input text. (3) There might be at most one error in between two words. We also assume that there might be at most one error after the last word.

We also use the following weak assumption: (4) We try to preserve the intended semantic meaning of the input text as much as possible.

[1] Whenever we use only the term '*word*' without an adjective (e.g., correct or incorrect), we imply a correct word.

2 Related Work

Some approaches consider spelling correction as text correction. An initial approach to automatic acquisition for context-based spelling correction was a statistical language-modeling approach using word and part-of-speech (POS) n-grams [2–5]. Some approaches in this paradigm use Bayesian classifiers and decision lists [6–8]. Other approaches simply focus on detecting sentences that contain errors, or computing a score that reflects the quality of the text [9–14].

In other text correction approaches, the prediction is typically framed as a classification task for a specific linguistic class, e.g., prepositions, near-synonym choices, or a set of predefined classes [15, 16]. In some approaches, a full syntactic analysis of the sentence is done to detect errors and propose corrections. We categorize this paradigm into two groups: those that constrain the rules of the grammar [17, 18], and those that use error-production rules [19–22].

[23] presents the use of a phrasal Statistical Machine Translation (SMT) techniques to identify and correct writing errors made by ESL (English as a Second Language) learners.

The work that is closely related to ours is that of Lee's [24], a supervised method built on the basic approach of template-matching on parse trees. To improve *recall*, the author uses the observed tree patterns for a set of verb form usages, and to improve *precision*, he utilizes n-grams as filters. [25] trains a maximum entropy model using lexical and POS features to recognize a variety of errors. Their evaluation data partially overlaps with that of [24] and our paper.

3 Proposed Method

Our proposed method determines some probable candidates and then sorts those candidates. We consider three similarity functions and one frequency value function in our method. One of the similarity functions, namely the string similarity function, is used to determine the candidate texts. The frequency value function and all the other similarity functions are used to sort the candidate texts.

3.1 Similarity and Frequency Value Functions

Similarity between Two Strings We use the same string similarity measure used in [26], with the following different normalization from [27]:

$$v_1 = \frac{2 \times len(LCS(s_1, s_2))}{len(s_1) + len(s_2)} \qquad v_2 = \frac{2 \times len(MCLCS_1(s_1, s_2))}{len(s_1) + len(s_2)}$$

$$v_3 = \frac{2 \times len(MCLCS_n(s_1, s_2))}{len(s_1) + len(s_2)} \qquad v_4 = \frac{2 \times len(MCLCS_z(s_1, s_2))}{len(s_1) + len(s_2)}$$

The similarity of the two strings, $S_1 \in [0, 1]$ is:

$$S_1(s_1, s_2) = \alpha_1 v_1 + \alpha_2 v_2 + \alpha_3 v_3 + \alpha_4 v_4 \tag{1}$$

Here, *len* calculates the length of a string, LCS, $MCLCS_1$, $MCLCS_n$, and $MCLCS_z$ calculate the *Longest Common Subsequence*, *Maximal Consecutive LCS* starting

at character 1, starting at character n, and ending at the last character between two strings, respectively. α_1, α_2, α_3, α_4 are weights and $\alpha_1+\alpha_2+\alpha_3+\alpha_4 = 1$. We heuristically set equal weights for most of our experiments[2].

Common Word Similarity between Texts If two texts have some words in common, we can measure their similarity based on the common words. We count the number of words in common between the text to correct and a candidate corrected text, normalizing the count by the size of both texts. Let us consider a pair of texts, T_1 and T_2 that have m and n tokens, with δ tokens in common. Thus, the common word similarity, $S_2 \in [0, 1]$ is:

$$S_2(T_1, T_2) = 2\delta/(m+n) \tag{2}$$

Non-Common Word Similarity If the two texts have some non-common words, we can measure how similar the two texts are based on their non-common words. If there are δ tokens in T_1 that exactly match with T_2, then there are $m-\delta$ and $n-\delta$ non-common words in texts T_1 and T_2, respectively, assuming that T_1 and T_2 have m and n tokens, respectively, and $n \geq m$. We remove all the δ common tokens from both T_1 and T_2. We construct a $(m-\delta)\times(n-\delta)$ *string similarity matrix* using Equation 1 and find out the maximum-valued matrix element. We add this matrix element to a list (say, ρ). We remove all the matrix elements which are in the row and column of the maximum-valued matrix element, from the original matrix. We remove the row and column, in order to remove the pair with maximum similarity. This makes the computation manageable: in the next steps, fewer words are left for matching. We repeat these steps until either the current maximum-valued matrix element is 0, or $m-\delta-|\rho| = 0$, or both. We sum up all the elements in ρ and divide by $n-\delta$ to get the non-common word similarity, $S_3 \in [0, 1)$:

$$S_3(T_1, T_2) = \textstyle\sum_{i=1}^{|\rho|} \rho_i/(n-\delta) \tag{3}$$

Normalized Frequency Value We determine the normalized frequency value of a candidate text (how we determine candidate texts is discussed in detail in Section 3.2) with respect to all other candidate texts. A candidate text having higher normalized frequency value is more likely a strong candidate for the correction, though not always. Let us consider, we have $\tilde{n}$ candidate texts for the input text T: $\{T_1, T_2, \cdots T_i \cdots, T_{\tilde{n}}\}$

$$\begin{aligned}
T_1 &= \{w_{11}, w_{12}, \cdots w_{1j} \cdots w_{(1)(m_1)}\} \\
T_2 &= \{w_{21}, w_{22}, \cdots w_{2j} \cdots w_{(2)(m_2)}\} \\
&\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\
T_i &= \{w_{i1}, w_{i2}, \cdots w_{ij} \cdots w_{(i)(m_i)}\} \\
&\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\
T_{\tilde{n}} &= \{w_{\tilde{n}1}, w_{\tilde{n}2}, \cdots w_{\tilde{n}j} \cdots w_{(\tilde{n})(m_{\tilde{n}})}\}
\end{aligned}$$

Here, w_{ij} is the jth token of the candidate text, T_i, and m_i means that the candidate text T_i has m_i tokens. It is important to note that the number of tokens

[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.

each candidate text has may be different from the rest. The number of 5-grams in any candidate text, T_i is $m_i - 4$. Again, let us consider that F_i is the set of frequencies of all the 5-grams that T_i has; f_{ij} is the frequency of the jth 5-gram of the candidate text, T_i. That is:

$$\begin{aligned} F_1 &= \{f_{11}, f_{12}, \cdots f_{1j} \cdots f_{(1)(m_1-4)}\} \\ F_2 &= \{f_{21}, f_{22}, \cdots f_{2j} \cdots f_{(2)(m_2-4)}\} \\ &\cdots\cdots\cdots\cdots\cdots\cdots \\ F_i &= \{f_{i1}, f_{i2}, \cdots f_{ij} \cdots f_{(i)(m_i-4)}\} \\ &\cdots\cdots\cdots\cdots\cdots\cdots \\ F_{\tilde{n}} &= \{f_{\tilde{n}1}, f_{\tilde{n}2}, \cdots f_{\tilde{n}j} \cdots f_{(\tilde{n})(m_{\tilde{n}}-4)}\} \end{aligned}$$

Here, $\{f_{11}, f_{21}, \cdots f_{i1} \cdots f_{\tilde{n}1}\}$, $\{f_{12}, f_{22}, \cdots f_{i2} \ \cdots f_{\tilde{n}2}\}$, $\{f_{1j}, f_{2j}, \cdots f_{ij} \cdots f_{\tilde{n}j}\}$ and $\{f_{(1)(m_i-4)}, f_{(2)(m_i-4)}, \cdots f_{(i)(m_i-4)} \cdots f_{(\tilde{n})(m_i-4)}\}$ are the sets of 5-gram frequencies for all $\tilde{n}$ candidate texts that are processed in the first step[3], the second step, the jth step, and the $(m_i - 4)$th step, respectively. We calculate the normalized frequency value of a candidate text as the summation of all the 5-gram frequencies of the candidate text over the summation of the maximum frequencies in each step that the candidate text may have. Thus the normalized frequency value of T_i represented as $S_4 \in [0, 1]$ is:

$$S_4(T_i) = \sum_{j=1}^{m_i-4} f_{ij} / \sum_{l=1}^{m_i-4} \max_{k \in N} f_{kl} \tag{4}$$

3.2 Determining Candidate Texts

Let us consider an input text, that after tokenization has m tokens, i.e., $T = \{w_1, w_2 \cdots, w_m\}$. Our approach consists in going from left to right according to a set of rules that are listed in Table 1 and Table 2. We use three basic operations, *Insert*, *Replace* and *Delete* to list these 5-gram rules. We also use *No Operation* to mean that we do not use any operation, rather we directly use the next token from T to list a 5-gram rule.

5-gram Rules Used in Step 1 Table 1 lists all possible 5-gram rules generated from the said operations and assumptions. We use each of these 5-gram rules to generate a set of 5-grams and their frequencies by trying to match the 5-gram rule with the Web 1T 5-grams. We take the decision of how many candidate 5-grams generated from each 5-gram rule we keep for further processing (say, $\bar{n}$). The 5-gram Rule #1 in Table 1 says that we take the first five tokens from T to generate a 5-gram and try to match with the Web 1T 5-grams to generate the only candidate 5-gram and its frequency, if there is any matching. In 5-gram Rule #2, we take the first four tokens from T and try to insert each word from a list of words (our goal here is to determine this list of words; it might be empty) in between w_1 and w_2 to generate a list of 5-grams and try to match with the

[3] By the first step, we mean the step when we process the first possible 5-grams in the input text. Similarly, by the second step, we mean the step when we process the next possible 5-grams (by removing the first token from the 5-grams used in first step and adding an extra word from the input text or other way, which is discussed in detail in Section 3.2) in the input text, and so on.

Web 1T 5-grams to generate a set of 5-grams and their frequencies. We sort these 5-grams in descending order by their frequencies and only keep at most the top $\bar{n}$ 5-grams and their frequencies. All I's and R's in Table 1 and Table 2 function similar to variables and all $w_i \in T$ function similar to constants. The 5-gram Rule #9 can generate a list of 5-grams and their frequencies, based on all the possible values of R_2, a set of all replaceable words of w_2. We determine the string similarity between w_2 and each member of R_2 using (1) and sort the list in descending order by string similarity values and only keep at most $\bar{n}$ 5-grams.

Table 1. List of all possible 5-gram rules in step 1

Rule#	5-gram Rule	Generated from	Rule#	5-gram Rule	Generated from
1	$w_1\ w_2\ w_3\ w_4\ w_5$	No Operation	26	$w_1\ w_3\ I_1\ w_4\ w_5$	
2	$w_1\ I_1\ w_2\ w_3\ w_4$		27	$w_1\ w_3\ w_4\ I_1\ w_5$	
3	$w_1\ w_2\ I_1\ w_3\ w_4$		28	$w_1\ w_2\ w_4\ I_1\ w_5$	Single Delete +
4	$w_1\ w_2\ w_3\ I_1\ w_4$	Single Insert	29	$w_1\ w_2\ w_4\ w_5\ I_1$	Single Insert
5	$w_1\ w_2\ w_3\ w_4\ I_1$		30	$w_1\ w_3\ w_4\ w_5\ I_1$	
6	$w_1\ I_1\ w_2\ I_2\ w_3$		31	$w_1\ w_2\ w_3\ w_5\ I_1$	
7	$w_1\ w_2\ I_1\ w_3\ I_2$	Double Insert	32	$w_1\ w_3\ w_4\ w_5\ R_6$	
8	$w_1\ I_1\ w_2\ w_3\ I_2$		33	$w_1\ w_2\ w_3\ w_5\ R_6$	
9	$w_1\ R_2\ w_3\ w_4\ w_5$		34	$w_1\ w_3\ R_4\ w_5\ w_6$	
10	$w_1\ w_2\ R_3\ w_4\ w_5$		35	$w_1\ w_2\ w_4\ R_5\ w_6$	Single Delete +
11	$w_1\ w_2\ w_3\ R_4\ w_5$	Single Replace	36	$w_1\ w_3\ w_4\ R_5\ w_6$	Single Replace
12	$w_1\ w_2\ w_3\ w_4\ R_5$		37	$w_1\ w_2\ w_4\ w_5\ R_6$	
13	$w_1\ R_2\ w_3\ R_4\ w_5$		38	$w_1\ R_2\ w_3\ w_5\ w_6$	
14	$w_1\ w_2\ R_3\ w_4\ R_5$	Double Replace	39	$w_1\ w_2\ R_3\ w_4\ w_6$	Single Replace +
15	$w_1\ R_2\ w_3\ w_4\ R_5$		40	$w_1\ R_2\ w_3\ w_4\ w_6$	Single Delete
16	$w_1\ w_3\ w_4\ w_5\ w_6$		41	$w_1\ I_1\ w_2\ R_3\ w_4$	
17	$w_1\ w_2\ w_4\ w_5\ w_6$		42	$w_1\ w_2\ I_1\ w_3\ R_4$	Single Insert +
18	$w_1\ w_2\ w_3\ w_5\ w_6$	Single Delete	43	$w_1\ I_1\ w_2\ w_3\ R_4$	Single Replace
19	$w_1\ w_2\ w_3\ w_4\ w_6$		44	$w_1\ R_2\ w_3\ I_1\ w_4$	Single Replace +
20	$w_1\ w_3\ w_5\ w_6\ w_7$		45	$w_1\ w_2\ R_3\ w_4\ I_1$	Single Insert
21	$w_1\ w_2\ w_4\ w_6\ w_7$		46	$w_1\ R_2\ w_3\ w_4\ I_1$	
22	$w_1\ w_3\ w_4\ w_6\ w_7$		47	$w_1\ I_1\ w_2\ w_4\ w_5$	
23	$w_1\ w_2\ w_4\ w_5\ w_7$	Double Delete	48	$w_1\ w_2\ I_1\ w_3\ w_5$	Single Insert +
24	$w_1\ w_2\ w_3\ w_5\ w_7$		49	$w_1\ I_1\ w_2\ w_3\ w_5$	Single Delete
25	$w_1\ w_3\ w_4\ w_5\ w_7$				

Limits for the Number of Steps, n_{tp} We figure out what maximum and minimum number of steps we need for an input text. Taking the second assumption into consideration, it is obvious that if the value of m is 3 (the number of words would also be 3) then only rules # 6, 7 and 8 can be used to generate 5-grams. 5-grams generated from rule # 7 and 8 can not be used in the next step as after the last word (w_3); we might have at most one error and all the 5-grams, if any, generated using these rules have this error (i.e., I_2). 5-grams generated from rules # 6 can be used in the next step (by rule # 2 in Table 2) to test whether we can insert a word in the next step, provided that the previous step generates at least one 5-gram. Thus, if $m = 3$ we might need at most 2 steps. Now, if $m = 4$, then for the added word (i.e., w_4) we need two extra steps to test rules # 5 and 2, in order, on top of the previous two steps (for the first three words), provided that each previous step generates at least one 5-gram.

That is, each extra token in T needs at most two extra steps. We generalize the maximum number of steps needed for an input text having m tokens as:

$$\text{Max } n_{tp} = 2+(m-3)\times 2 = 2m-4 \quad (5)$$

Again, the minimum number of steps is ensured if rules # 6 to 8 in step 1 do not generate any 5-gram. This means that, if $m = 3$, we might need at least 0 steps[4]. Now, if $m = 4$ then for the added word (i.e., w_4) we need only an extra step to test rule # 5 on top of the previous single step (for the first three tokens). That is, each extra token in T needs at least one extra step, provided that each previous step for each extra token generates at least one 5-gram.[5] We generalize the minimum number of steps needed for an input text having m tokens as:

$$\text{Min } n_{tp} = m-3 \quad (6)$$

In (5), the maximum number of steps, $2m-4$, also means that the maximum number of tokens possible in a candidate text is $2m$. Thus, an input text having $2m$ tokens can have at most m errors to be handled and m correct words, assuming $m \geq 3$ (the second assumption on page 1).

Table 2. List of All Possible 5-gram Rules in Step 2 to Step $2m-4$

Rule#	5-gram Rule	Generated from	Case Number
1	$---w_iw_{i+1}$	No Operation	
2	$---w_iI_j$	Single Insert	1: if the last word
3	$---w_iw_{i+2}$	Single Delete	in step 1 is in T
4	$---w_iR_{i+1}$	Single Replace	
5	$--w_iI_jw_{i+1}$	No Operation	2: if the second last word in step 1 is in
6	$--w_iR_{i+1}w_{i+2}$	No operation	T and the last word is either an inserted or a replaced word

5-gram Rules used in Step 2 to $2m-4$ Table 2 lists all possible 5-gram rules generated from the said operations and assumptions for step 2 to step $2m-4$. We use step 2 (i.e., the next step) only if step 1 (i.e., the previous step) generates at least one 5-gram from 5-gram rules listed in Table 1. Similarly, we use step 3 (i.e., the next step) only if step 2 (i.e., the previous step) generates at least one 5-gram from the 5-gram rules listed in Table 2, and so on. In Table 2, '$-$' means that it might be any word that is in T, or an inserted word (an instance of I's), or a replaced word (an instance of R's) in the previous step. To give a specific example of how we list the 5-gram rules in Table 2, consider that rule #2 (w_1 I_1 w_2 w_3 w_4) in Table 1 generates at least one 5-gram in step 1. We take the last four words of this 5-gram (i.e., I_1 w_2 w_3 w_4) and add the next word from T (in this case w_5), in order to form a new rule in step 2 (which is I_1 w_2 w_3 w_4 w_5). The general form of this rule ($---w_i\,w_{i+1}$) is listed as rule #1 in Table 2. In step 1, I_1 in rule #2 acts like a variable, but in step 2 we use only a single instance of I_1, which acts like a constant. We categorize all

[4] We call a step successful if it generates at least one 5-gram. Thus, if we try to generate some 5-grams in step 1 and if we fail to generate any, then the number of step, n_{tp} is 0, though we do some processing for step 1.

[5] If we omit the assumption that each previous step for each extra token generates at least one 5-gram, then to determine the Min n_{tp} is very straight forward, it is 0.

the 5-grams generated in step 1 (i.e., the previous step) into two different cases. Case 1 groups each 5-gram in step 1 having its last word in T. Case 2 groups each 5-gram in step 1 having its second last word in T, and the last word not in T. We stop when we fail to generate any 5-gram in the next step from all the 5-gram rules of the previous step.

Determining the Limit of Candidate Texts There might be a case when no 5-gram is generated in step 1; this means that the minimum $\tilde{n}$ possible is 0. Table 1 shows that there are 11 5-gram rules (rules without any I's or R's) in step 1 that generate at most one 5-gram per 5-gram rule. It turns out that the remaining 5-gram rules can generate at most $\bar{n}$ 5-grams per 5-gram rule. Thus, the maximum number of candidate texts, $\tilde{n}$, that can be generated having only a single step (i.e., $n_{tp} = 1$) is:

$$\begin{aligned}\text{Max } \tilde{n} = &(\text{no. of 5-gram rules in step 1} - \text{no. of 5-gram rules in step 1 without} \\ &\text{any } I\text{'s or } R\text{'s}) \times \bar{n} + \text{no. of 5-gram rules in step 1 without any } I\text{'s or } R\text{'s} \quad (7)\\ = &(49 - 11) \times \bar{n} + 11 = 38\bar{n} + 11 \quad (8)\end{aligned}$$

At most $2\bar{n} + 2$ 5-grams (rules #1 to 4 in Table 2) can be generated in step 2 from a single 5-gram generated in step 1 having the last word in T. There may be at most 33 such 5-grams in step 1. At most 1 5-gram (rules #5 and 6 in Table 2) can be generated in step 2 from a single 5-gram generated in step 1 having the second last word in T and the last word being either an inserted or a replaced word. There may be at most 16 such 5-grams in step 1. The maximum number of candidate texts, $\tilde{n}$, that can be generated having two steps (i.e., $n_{tp} = 2$) is:

$$\text{Max } \tilde{n} = 33(2\bar{n} + 2) + 16 \times 1 \quad (9)$$

We generalize Max $\tilde{n}$ for different values of n_{tp} as:

$$\text{Max } \tilde{n} \approx \begin{cases} 38\bar{n}+11 & \text{if step} = 1 \\ 33\times 2^0(2\bar{n}+2)+16 & \text{if step} = 2 \\ 33\times 2^1(2\bar{n}+2)+66\times 2^0\bar{n} & \text{if step} = 3 \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots & \\ 33\times 2^{n_{tp}-2}(2\bar{n}+2)+66\times 2^{n_{tp}-3}\bar{n} & \text{if step} = n_{tp} \end{cases} \quad (10)$$

Simplifying (10):

$$\text{Max } \tilde{n} \approx \begin{cases} 38\bar{n}+11 & \text{if step} = 1 \\ 66\bar{n}+82 & \text{if step} = 2 \\ 2^{n_{tp}-3}(198\bar{n}+132) & \text{if step} \geq 3 \end{cases} \quad (11)$$

Theoretically, Max $\tilde{n}$ seems to be a large number, but practically $\tilde{n}$ is much smaller than Max $\tilde{n}$. This is because not all theoretically possible 5-grams are in the Web 1T 5-grams data set, and because fewer 5-grams generated in any step have an effect in all the subsequent steps.

Forming Candidate Texts Algorithm 1 describes how a list of candidate texts can be formed from the list of 5-grams in each step. That is, the output of

Algorithm 1: forming candidate texts

```
   input  : n_tp, list of 5-grams in each step
   output: candidate_list
 1 candidate_list ← NULL
 2 for each 5-gram of step 1 do
 3 |  k ← 1
 4 |  candidate_text[k] ← 5-gram of step 1
 5 |  for i ← 2 to n_tp do
 6 |  |  j ← 1
 7 |  |  for each k do
 8 |  |  |  for each 5-gram of step i do
 9 |  |  |  |  next_5-gram← 5-gram of step i
10 |  |  |  |  temp_candidate_text[j] ← candidate_text[k]
11 |  |  |  |  str_1 ← last four words of  temp_candidate_text[j]
12 |  |  |  |  str_2 ← first four words of  next_5-gram
13 |  |  |  |  if str_1 = str_2 then
14 |  |  |  |  |  temp_candidate_text[j] ← temp_candidate_text[j] . last
   |  |  |  |  |  word of next_5-gram   /* '.' is to concatenate */
15 |  |  |  |  end
16 |  |  |  |  increment j
17 |  |  |  end
18 |  |  end
19 |  |  decrement j
20 |  |  for each j do
21 |  |  |  candidate_text[j] ← temp_candidate_text[j]
22 |  |  end
23 |  |  k ← j
24 |  end
25 |  for each k do
26 |  |  candidate_list ← candidate_list + candidate_text[k]
27 |  end
28 end
```

Algorithm 1 is $\{T_1, T_2, \cdots T_i \cdots, T_{\tilde{n}}\}$. The algorithm works as follows: Taking the last four words of each 5-gram in step 1, it tries to match with the first four words of each 5-gram in step 2. If it matches, then concatenating the last word of the matched 5-gram in step 2 with the matched 5-gram in step 1 generates a temporary candidate text for further processing. If a 5-gram in step 1 does not match with at least a single 5-gram in step 2, then the 5-gram in step 1 is a candidate text. One 5-gram in step 1 can match with several 5-grams in step 2, thus generating several temporary candidate texts. We continue this process until we cover all the steps.

3.3 Sorting Candidate Texts

It turns out from 3.2 that, if the input text is T, then the total $\tilde{n}$ candidate texts are $\{T_1, T_2, \cdots T_i \cdots, T_{\tilde{n}}\}$. We determine the correctness value, S for each candidate text using (12), a weighted sum of (2), (3) and (4), and then we sort in descending order by the correctness values. In (12), it is obvious that $\beta_1 + \beta_2 + \beta_3 = 1$ to have $S \in (0, 1]$.

$$S(T_i)=\beta_1 S_2(T_i,T)+\beta_2 S_3(T_i,T)+\beta_3 S_4(T_i) \quad (12)$$

By trying to preserve the semantic meaning of the input text as much as possible, we intentionally keep the candidate texts and the input text as close (both semantically and syntactically) as possible. Thus, we set more weight on S_2 and S_3. Though we set low weight on S_4, it is one of the most crucial parts of the method, that helps to identify and correct the error. If we only rely on the normalized frequency value of each candidate text, then we have to deal with an increasing number of *false positives*: the method detects an input text as incorrect, while, in reality, it is not. On the contrary, if we only rely on the similarity of common words, non-common words, and so on, between input text and each candidate text, then we have to deal with an increasing number of *false negatives*: the method detects an input text as correct, while, in reality, it is not.

4 Evaluation and Experimental Results

4.1 Evaluation on WSJ Corpus

Because of the lack of a publicly available data set having multiple errors in short texts, we generate a new evaluation data set, utilizing the 1987-89 *Wall Street Journal* corpus. It is assumed that this data contains no errors. We select 34 short texts from this corpus and artificially introduce some errors, so that it requires to perform some combinations of *insert*, and/or *delete*, and/or *replace* operation to get back to the correct texts. To generate the incorrect texts, we artificially insert prepositions and articles, delete articles, prepositions, auxiliary verbs, and replace prepositions with other prepositions, singular nouns with plural nouns (e.g., *spokesman* with *spokesmen*), articles with other articles, real words with real-word spelling errors (e.g., *year* with *tear*), real words with spelling errors (e.g., *there* with *ther*). To generate real-word spelling errors (which are in fact semantic errors) and spelling errors, we use the same procedure as [28]. The average number of tokens in a correct text and an incorrect text are 7.44 and 6.32, respectively. The average number of corrections required per text is 1.76. We keep some texts without inserting any error, to test the robustness of the system (we got only a single false positive). This decreases the number of errors per text.

The performance is measured using *Recall* (R), *Precision* (P), F_1 and *Accuracy* (*Acc*). We asked two human judges, both native speakers of English and graduate students in Natural Language Processing, to correct those 34 texts. The agreement between the two judges is low (the detection agreement is 53.85% and the correction agreement is 50.77%), which means the task is difficult even for human experts. Table 3 shows two examples of test texts. The results in Table 4 show that our method gives comparable *recall* value for both detection and correction, whereas human judges give better *precision* value for both detection and correction. Since a majority of the words in the evaluation data set are correct, the *baseline* is to propose no correction, achieving 76.28% accuracy. Taking this *baseline* accuracy as a lower limit and the accuracy achieved by the human

Table 3. Some examples.

	Example 1	Example 2
Incorrect	All funding decisions is made the	What his believed to the next
Correct	All funding decisions are made by the	What is believed to be the next
Judge 1	All funding decisions is made by the	What is believed to be next
Judge 2	All funding decisions are made the	What he believed to be the next
Our Method	All funding decisions are made by the	What is believed to be the next

Table 4. Results on the WSJ corpus.

	Detection			**Correction**			**Acc.**
	R	**P**	**F_1**	**R**	**P**	**F_1**	
Our Method	90.0	75.00	81.82	78.33	65.28	71.21	84.98
Judge 1	65.0	88.64	75.00	58.33	79.54	67.31	86.56
Judge 2	90.0	93.10	91.53	83.33	86.21	84.75	92.89

judges as an upper limit, we conclude that the automatic method realizes about half of the possible improvement between the baseline and the human expert upper bound (76%-84%-92%, respectively).

4.2 Evaluation on JLE Corpus

We also evaluate the proposed method using the NICT JLE corpus [22], to directly compare with [24]. The JLE corpus has 15,637 sentences with annotated grammatical errors and their corrections. We generated a test set of 477 sentences for subject-verb (S-V) agreement errors, and another test set of 238 sentences for auxiliary agreement and complementation (AAC) errors by retaining the verb form errors, but correcting all other error types. [24] generated the same number of sentences of each category.

[24] used the *majority baseline*, which is to propose no correction, since the vast majority of verbs were in their correct forms. Thus, [24] achieved a *majority baseline* of 96.95% for S-V agreement and 98.47% for AAC. Based on these numbers, it can be determined that [24] had only 14 or 15 errors in the S-V agreement data set and 3 or 4 errors in the AAC data set. Our data set has a *majority baseline* of 80.5% for S-V agreement and 79.8% for AAC. It means that we have 93 errors in the S-V agreement data set and 48 errors in the AAC data set. The small number of errors in their data set is the reason why they get high accuracy even when they have moderate *precision* and *recall*. For example, if their method fails to correct 2 errors out of the 3 errors in the S-V agreement data set (i.e., if true positive is 1 and false positives are 2), then their *recall* would be 33.3%, even then their accuracy would be 99.16%. Table 5 shows that our method generates consistent *precision*, *recall*, and accuracy.

5 Conclusion

The proposed unsupervised text correction approach can correct one error, which might be syntactic or semantic, for every word in a text. This large magnitude of error coverage, in terms of number, can be applied to correct Optical Character

Table 5. Results on the JLE corpus.'—' means that the result is not mentioned in [24].

	Detection			Correction			Acc.
	R	**P**	**F_1**	**R**	**P**	**F_1**	
Lee (S-V)	—	83.93	—	80.92	81.61	—	98.93
Lee (AAC)	—	80.67	—	42.86	68.0	—	98.94
Our (S-V)	98.92	96.84	97.87	97.85	95.79	96.81	98.74
Our (AAC)	97.92	94.0	95.92	95.83	92.0	93.88	97.48

Recognition (OCR) errors, to automatically-mark (based on grammar and semantics) subjective examination papers, etc. A major drawback of our proposed approach is the dependence on the availability of enough 5-grams. The future challenge is how to tackle this problem, while keeping the approach unsupervised.

References

1. Brants, T., Franz, A.: Web 1T 5-gram corpus version 1.1. Technical report, Google Research (2006)
2. Atwell, E., Elliot, S.: Dealing with ill-formed english text. In Garside, R., Sampson, G., Leech, G., eds.: The computational analysis of English: a corpus-based approach, London, Longman (1987)
3. Gale, W.A., Church, K.W.: Estimation procedures for language context: Poor estimates are worse than none. In: Proceedings Computational Statistics, Physica-Verlag, Heidelberg (1990) 69–74
4. Mays, E., Damerau, F.J., Mercer, R.L.: Context based spelling correction. Information Processing and Management **27**(5) (1991) 517–522
5. Church, K.W., Gale, W.A.: Probability scoring for spelling correction. Statistics and Computing **1**(2) (December 1991) 93–103
6. Golding, A.R., Roth, D.: A winnow-based approach to context-sensitive spelling correction. Machine Learning **34**(1-3) (1999) 107–130
7. Golding, A.R., Schabes, Y.: Combining trigram-based and feature-based methods for context-sensitive spelling correction. In: Proceedings of the 34th annual meeting on Association for Computational Linguistics, Morristown, NJ, USA, Association for Computational Linguistics (1996) 71–78
8. Yarowsky, D.: Decision lists for lexical ambiguity resolution: application to accent restoration in spanish and french. In: Proceedings of the 32nd annual meeting on Association for Computational Linguistics, Morristown, NJ, USA, Association for Computational Linguistics (1994) 88–95
9. Gamon, M., Aue, A., Smets, M.: Sentence-level mt evaluation without reference translations: Beyond language modeling. In: European Association for Machine Translation (EAMT). (2005) 103–111
10. Sjöbergh, J.: Chunking: an unsupervised method to find errors in text. In Werner, S., ed.: Proceedings of the 15th NoDaLiDa conference. (2005) 180–185
11. Wang, C., Seneff, S.: High-quality speech translation for language learning. In: Proc. of InSTIL, Venice, Italy (2004)
12. Eeg-olofsson, J., Knutsson, O.: Automatic grammar checking for second language learners - the use of prepositions. In: NoDaLiDa, Reykjavik, Iceland (2003)
13. Chodorow, M., Leacock, C.: An unsupervised method for detecting grammatical errors. In: Proceedings of NAACL'00. (2000) 140–147

14. Atwell, E.S.: How to detect grammatical errors in a text without parsing it. In: Proceedings of the third conference on European chapter of the Association for Computational Linguistics, Morristown, NJ, USA, Association for Computational Linguistics (1987) 38–45
15. Islam, A., Inkpen, D.: An unsupervised approach to preposition error correction. In: Proceedings of the IEEE International Conference on Natural Language Processing and Knowledge Engineering (IEEE NLP-KE'10), Beijing (August 2010) 1–4
16. Felice, R.D., Pulman, S.G.: A classifier-based approach to preposition and determiner error correction in L2 English. In: Proceedings of the 22nd International Conference on Computational Linguistics (Coling 2008), Manchester, UK, Coling 2008 Organizing Committee (August 2008) 169–176
17. Fouvry, F.: Constraint relaxation with weighted feature structures. In: Proceedings of the 8th International Workshop on Parsing Technologies, Nancy, France (2003) 23–25
18. Vogel, C., Cooper, R.: Robust chart parsing with mildly inconsistent feature structures. In Schter, A., Vogel, C., eds.: Nonclassical Feature Systems. Volume 10. Centre for Cognitive Science, University of Edinburgh (1995) Working Papers in Cognitive Science.
19. Wagner, J., Foster, J., van Genabith, J.: A comparative evaluation of deep and shallow approaches to the automatic detection of common grammatical errors. In: Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL). (2007) 112–121
20. Andersen, O.E.: Grammatical error detection using corpora and supervised learning. In Nurmi, V., Sustretov, D., eds.: Proceedings of the 12th Student Session of the European Summer School for Logic, Language and Information. (2007)
21. Foster, J., Vogel, C.: Parsing ill-formed text using an error grammar. Artif. Intell. Rev. **21**(3-4) (2004) 269–291
22. Izumia, E., Uchimotoa, K., Isaharaa, H.: SST speech corpus of Japanese learners' English and automatic detection of learners' errors. ICAME Journal **28** (2004) 31–48
23. Brockett, C., Dolan, W.B., Gamon, M.: Correcting ESL errors using phrasal SMT techniques. In: ACL-44: Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics, Morristown, NJ, USA, Association for Computational Linguistics (2006) 249–256
24. Lee, J.S.Y.: Automatic Correction of Grammatical Errors in Non-native English Text. PhD thesis, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science (June 2009)
25. Izumi, E., Supnithi, T., Uchimoto, K., Isahara, H., Saiga, T.: Automatic error detection in the japanese learners english spoken data. In: In Companion Volume to Proc. ACL03. (2003) 145–148
26. Islam, A., Inkpen, D.: Real-word spelling correction using Google Web 1T n-gram data set. In Cheung, D.W.L., Song, I.Y., Chu, W.W., Hu, X., Lin, J.J., eds.: Proceedings of the 18th ACM Conference on Information and Knowledge Management, CIKM 2009, Hong Kong, ACM (November 2009) 1689–1692
27. Islam, A., Inkpen, D.: Semantic text similarity using corpus-based word similarity and string similarity. ACM Trans. Knowl. Discov. Data **2** (July 2008) 10:1–10:25
28. Hirst, G., Budanitsky, A.: Correcting real-word spelling errors by restoring lexical cohesion. Natural Language Engineering **11**(1) (March 2005) 87–111