Better Evaluation for Grammatical Error Correction

Daniel Dahlmeier[1] and **Hwee Tou Ng**[1,2]
[1]NUS Graduate School for Integrative Sciences and Engineering
[2]Department of Computer Science, National University of Singapore
{danielhe,nght}@comp.nus.edu.sg

Abstract

We present a novel method for evaluating grammatical error correction. The core of our method, which we call *MaxMatch* (M^2), is an algorithm for efficiently computing the sequence of phrase-level edits between a source sentence and a system hypothesis that achieves the highest overlap with the gold-standard annotation. This optimal edit sequence is subsequently scored using F_1 measure. We test our M^2 scorer on the Helping Our Own (HOO) shared task data and show that our method results in more accurate evaluation for grammatical error correction.

1 Introduction

Progress in natural language processing (NLP) research is driven and measured by automatic evaluation methods. Automatic evaluation allows fast and inexpensive feedback during development, and objective and reproducible evaluation during testing time. Grammatical error correction is an important NLP task with useful applications for second language learning. Evaluation for error correction is typically done by computing F_1 measure between a set of proposed system edits and a set of human-annotated gold-standard edits (Leacock et al., 2010).

Unfortunately, evaluation is complicated by the fact that the set of edit operations for a given system hypothesis is ambiguous. This is due to two reasons. First, the set of edits that transforms one string into another is not necessarily unique, even at the token level. Second, edits can consist of longer phrases which introduce additional ambiguity. To see how this can affect evaluation, consider the following source sentence and system hypothesis from the recent Helping Our Own (HOO) shared task (Dale and Kilgarriff, 2011) on grammatical error correction:

Source :	Our baseline system feeds word into PB-SMT pipeline.
Hypot. :	Our baseline system feeds **a** word into PB-SMT pipeline.

The HOO evaluation script extracts the system edit ($\epsilon \rightarrow$ a), i.e., inserting the article *a*. Unfortunately, the gold-standard annotation instead contains the edits (word $\rightarrow$ {a word, words}). Although the extracted system edit results in the *same* corrected sentence as the first gold-standard edit option, the system hypothesis was considered to be invalid.

In this work, we propose a method, called *MaxMatch* (M^2), to overcome this problem. The key idea is that if there are multiple possible ways to arrive at the same correction, the system should be evaluated according to the set of edits that matches the gold-standard as often as possible. To this end, we propose an algorithm for efficiently computing the set of phrase-level edits with the maximum overlap with the gold standard. The edits are subsequently scored using F_1 measure. We test our method in the context of the HOO shared task and show that our method results in a more accurate evaluation for error correction.

The remainder of this paper is organized as follows: Section 2 describes the proposed method; Section 3 presents experimental results; Section 4 discusses some details of grammar correction evaluation; and Section 5 concludes the paper.

2 Method

We begin by establishing some notation. We consider a set of *source sentences* $S = \{\mathbf{s}_1, \ldots, \mathbf{s}_n\}$ together with a set of *hypotheses* $H = \{\mathbf{h}_1, \ldots, \mathbf{h}_n\}$ generated by an error correction system. Let $G = \{\mathbf{g}_1, \ldots, \mathbf{g}_n\}$ be the set of gold standard annotations for the same sentences. Each annotation $\mathbf{g}_i = \{g_i^1, \ldots, g_i^r\}$ is a set of *edits*. An edit is a triple (a, b, C), consisting of:

- start and end (token-) offsets a and b with respect to a source sentence,
- a correction C. For gold-standard edits, C is a set containing one or more possible corrections. For system edits, C is a single correction.

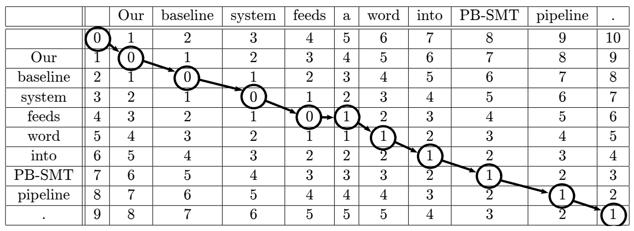
Evaluation of the system output involves the following two steps:

1. Extracting a set of *system edits* $\mathbf{e}_i$ for each source-hypothesis pair $(\mathbf{s}_i, \mathbf{h}_i)$.
2. Evaluating the system edits for the complete test set with respect to the gold standard G.

The remainder of this section describes a method for solving these two steps. We start by describing how to construct an *edit lattice* from a source-hypothesis pair. Then, we show that finding the optimal sequence of edits is equivalent to solving a shortest path search through the lattice. Finally, we describe how to evaluate the edits using F_1 measure.

2.1 Edit lattice

We start from the well-established Levenshtein distance (Levenshtein, 1966), which is defined as the minimum number of insertions, deletions, and substitutions needed to transform one string into another. The Levenshtein distance between a source sentence $\mathbf{s}_i = s_i^1, \ldots, s_i^k$ and a hypothesis $\mathbf{h}_i = h_i^1, \ldots, h_i^l$ can be efficiently computed using a two dimensional matrix that is filled using a classic dynamic programming algorithm. We assume that both $\mathbf{s}_i$ and $\mathbf{h}_i$ have been tokenized. The matrix for the example from Section 1 is shown in Figure 1. By performing a simple breadth-first search, similar to the Viterbi algorithm, we can extract the lattice of all shortest paths that lead from the top-left corner to the bottom-right corner of the Levenshtein matrix. Each vertex in the lattice corresponds to a cell



		Our	baseline	system	feeds	a	word	into	PB-SMT	pipeline	.
	0	1	2	3	4	5	6	7	8	9	10
Our	1	0	1	2	3	4	5	6	7	8	9
baseline	2	1	0	1	2	3	4	5	6	7	8
system	3	2	1	0	1	2	3	4	5	6	7
feeds	4	3	2	1	0	1	2	3	4	5	6
word	5	4	3	2	1	1	1	2	3	4	5
into	6	5	4	3	2	2	2	1	2	3	4
PB-SMT	7	6	5	4	3	3	3	2	1	2	3
pipeline	8	7	6	5	4	4	4	3	2	1	2
.	9	8	7	6	5	5	5	4	3	2	1

Figure 1: The Levenshtein matrix and the shortest path for a source sentence "Our baseline system feeds word into PB-SMT pipeline ." and a hypothesis "Our baseline system feeds a word into PB-SMT pipeline ."

in the Levenshtein matrix, and each edge in the lattice corresponds to an atomic edit operation: inserting a token, deleting a token, substituting a token, or leaving a token unchanged. Each path through the lattice corresponds to a shortest sequence of edits that transform $\mathbf{s}_i$ into $\mathbf{h}_i$. We assign a unit cost to each edge in the lattice.

We have seen that annotators can use longer phrases and that phrases can include unchanged words from the context, e.g., the gold edit from the example in Section 1 is $(4, 5, \text{word}, \{\text{a word, words}\})$. However, it seems unrealistic to allow an arbitrary number of unchanged words in an edit. In particular, we want to avoid very large edits that cover complete sentences. Therefore, we limit the number of unchanged words by a parameter u. To allow for phrase-level edits, we add transitive edges to the lattice as long as the number of unchanged words in the newly added edit is not greater than u and the edit changes at least one word. Let $e_1 = (a_1, b_1, C_1)$ and $e_2 = (a_2, b_2, C_2)$ be two edits corresponding to adjacent edges in the lattice, with the first end offset b_1 being equal to the second start offset a_2. We can combine them into a new edit $e_3 = (a_1, b_2, C_1 + C_2)$, where $C_1 + C_2$ is the concatenation of strings C_1 and C_2. The cost of a transitive edge is the sum of the costs of its parts. The lattice extracted from the example sentence is shown in Figure 2.

2.2 Finding maximally matching edit sequence

Our goal is to find the sequence of edits $\mathbf{e}_i$ with the maximum overlap with the gold standard. Let $L = (V, E)$ be the edit lattice graph from the last section. We change the cost of each edge whose cor-

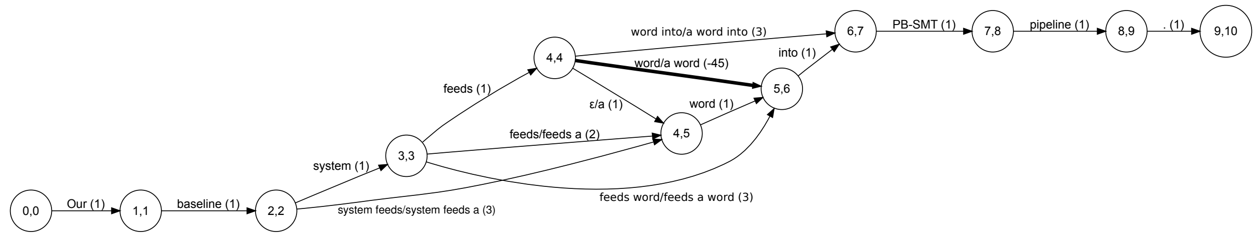


Figure 2: The edit lattice for "Our baseline system feeds ($\epsilon \rightarrow$ a) word into PB-SMT pipeline ." Edge costs are shown in parentheses. The edge from (4,4) to (5,6) matches the gold annotation and carries a negative cost.

responding edit has a match in the gold standard to $-(u+1) \times |E|$. An edit e *matches* a gold edit g iff they have the same offsets and e's correction is included in g:

$$match(e, g) \Leftrightarrow e.a = g.a \wedge e.b = g.b \wedge e.C \in g.C \quad (1)$$

Then, we perform a single-source shortest path search with negative edge weights from the start to the end vertex[1]. This can be done efficiently, for example with the Bellman-Ford algorithm (Cormen et al., 2001). As the lattice is acyclic, the algorithm is guaranteed to terminate and return a shortest path.

Theorem 1. *The set of edits corresponding to the shortest path has the maximum overlap with the gold standard annotation.*

Proof. Let $\mathbf{e} = e^1, \ldots, e^k$ be the edit sequence corresponding to the shortest path and let p be the number of matched edits. Assume that there exists another edit sequence $\mathbf{e}'$ with higher total edge weights but $p' > p$ matching edits. Then we have

$$\begin{aligned} p(-(u+1)|E|) + q &\leq p'(-(u+1)|E|) + q' \quad (2) \\ \Leftrightarrow (q - q') &\leq (p' - p)(-(u+1)|E|), \end{aligned}$$

where q and q' denote the combined cost of all non-matching edits in the two paths, respectively. Because $p' - p \geq 1$, the right hand side is at most $-(u+1)|E|$. Because q and q' are positive and bounded by $(u+1)|E|$, the left hand side cannot be smaller than or equal to $-(u+1)|E|$. This is a contradiction. Therefore there cannot exist such an edit sequence $\mathbf{e}'$, and $\mathbf{e}$ is the sequence with the maximum overlap with the gold-standard annotation. □

2.3 Evaluating edits

What is left to do is to evaluate the set of edits with respect to the gold standard. This is done by computing precision, recall, and F_1 measure (van Rijsbergen, 1979) between the set of system edits $\{\mathbf{e}_1, \ldots, \mathbf{e}_n\}$ and the set of gold edits $\{\mathbf{g}_1, \ldots, \mathbf{g}_n\}$ for all sentences

$$\begin{aligned} P &= \frac{\sum_{i=1}^{n} |\mathbf{e}_i \cap \mathbf{g}_i|}{\sum_{i=1}^{n} |\mathbf{e}_i|} && (3) \\ R &= \frac{\sum_{i=1}^{n} |\mathbf{e}_i \cap \mathbf{g}_i|}{\sum_{i=1}^{n} |\mathbf{g}_i|} && (4) \\ F_1 &= 2 \times \frac{P \times R}{P + R}, && (5) \end{aligned}$$

where we define the intersection between $\mathbf{e}_i$ and $\mathbf{g}_i$ as

$$\mathbf{e}_i \cap \mathbf{g}_i = \{e \in \mathbf{e}_i \mid \exists\, g \in \mathbf{g}_i(match(e, g))\}. \quad (6)$$

3 Experiments and Results

We experimentally test our M^2 method in the context of the HOO shared task. The HOO test data[2] consists of text fragments from NLP papers together with manually-created gold-standard corrections (see (Dale and Kilgarriff, 2011) for details). We test our method by re-scoring the best runs of the participating teams[3] in the HOO shared task with our M^2 scorer and comparing the scores with the official HOO scorer, which simply uses GNU `wdiff`[4] to extract system edits. We obtain each system's output and segment it at the sentence level according to the gold standard sentence segmentation. The

[1]To break ties between non-matching edges, we add a small cost $\zeta \ll 1$ to all non-matching edges, thus favoring paths that use fewer edges, everything else being equal.

[2]Available at http://groups.google.com/group/hoo-nlp/ after registration.

[3]Except one team that did not submit any plain text output.

[4]http://www.gnu.org/s/wdiff/

M^2 scorer	. . . should basic translational unit be (word $\rightarrow$ a word) . . .
HOO scorer	. . . should basic translational unit be *($\epsilon \rightarrow$ a) word . . .
M^2 scorer	. . . development set similar (with $\rightarrow$ to) ($\epsilon \rightarrow$ the) test set . . .
HOO scorer	. . . development set similar *(with $\rightarrow$ to the) test set . . .
M^2 scorer	($\epsilon \rightarrow$ The) *(Xinhua portion of $\rightarrow$ xinhua portion of) the English Gigaword3 . . .
HOO scorer	*(Xinhua $\rightarrow$ The xinhua) portion of the English Gigaword3 . . .

Table 2: Examples of different edits extracted by the M^2 scorer and the official HOO scorer. Edits that do not match the gold-standard annotation are marked with an asterisk (*).

Team	HOO scorer			M^2 scorer		
	P	R	F_1	P	R	F_1
JU (0)	10.39	3.78	5.54	12.30	4.45	6.53
LI (8)	20.86	3.22	5.57	21.12	3.22	5.58
NU (0)	29.10	7.38	11.77	31.09	7.85	12.54
UI (1)	50.72	13.34	21.12	54.61	14.57	23.00
UT (1)	5.01	4.07	4.49	5.72	4.45	5.01

Table 1: Results for participants in the HOO shared task. The run of the system is shown in parentheses.

source sentences, system hypotheses, and corrections are tokenized using the Penn Treebank standard (Marcus et al., 1993). The character edit offsets are automatically converted to token offsets. We set the parameter u to 2, allowing up to two unchanged words per edit. The results are shown in Table 1. Note that the M^2 scorer and the HOO scorer adhere to the same score definition and only differ in the way the system edits are computed. We can see that the M^2 scorer results in higher scores than the official scorer for all systems, showing that the official scorer missed some valid edits. For example, the M^2 scorer finds 155 valid edits for the UI system compared to 141 found by the official scorer, and 83 valid edits for the NU system, compared to 78 by the official scorer. We manually inspect the output of the scorers and find that the M^2 scorer indeed extracts the correct edits matching the gold standard where possible. Examples are shown in Table 2.

4 Discussion

The evaluation framework proposed in this work differs slightly from the one in the HOO shared task.

Sentence-by-sentence. We compute the edits between source-hypothesis sentence pairs, while the HOO scorer computes edits at the document level. As the HOO data comes in a sentence-segmented format, both approaches are equivalent, while sentence-by-sentence is easier to work with.

Token-level offsets. In our work, the start and end of an edit are given as *token offsets*, while the HOO data uses character offsets. Character offsets make the evaluation procedure very brittle as a small change, e.g., an additional whitespace character, will affect all subsequent edits. Character offsets also introduce ambiguities in the annotation, e.g., whether a comma is part of the preceding token.

Alternative scoring. The HOO shared task defines three different scores: *detection*, *recognition*, and *correction*. Effectively, all three scores are F_1 measures and only differ in the conditions on when an edit is counted as valid. Additionally, each score is reported under a "with bonus" alternative, where a system receives rewards for missed optional edits. The F_1 measure defined in Section 2.3 is equivalent to *correction without bonus*. Our method can be used to compute detection and recognition scores and scores with bonus as well.

5 Conclusion

We have presented a novel method, called Max-Match (M^2), for evaluating grammatical error correction. Our method computes the sequence of phrase-level edits that achieves the highest overlap with the gold-standard annotation. Experiments on the HOO data show that our method overcomes deficiencies in the current evaluation method. The M^2 scorer is available for download at http://nlp.comp.nus.edu.sg/software/.

Acknowledgments

We thank Chang Liu for comments on an earlier draft. This research is supported by the Singapore National Research Foundation under its International Research Centre @ Singapore Funding Initiative and administered by the IDM Programme Office.

References

T. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. 2001. *Introduction to Algorithms*. MIT Press, Cambridge, MA.

R. Dale and A. Kilgarriff. 2011. Helping Our Own: The HOO 2011 pilot shared task. In *Proceedings of the 2011 European Workshop on Natural Language Generation*.

C. Leacock, M. Chodorow, M. Gamon, and J. Tetreault, 2010. *Automated Grammatical Error Detection for Language Learners*, chapter 5. Morgan and Claypool Publishers.

V. Levenshtein. 1966. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10(8):707–710.

M.P. Marcus, M.A. Marcinkiewicz, and B. Santorini. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19.

C. J. van Rijsbergen. 1979. *Information Retrieval*. Butterworth, 2nd edition.