Improved Correction Detection in Revised ESL Sentences

Huichao Xue and **Rebecca Hwa**
Department of Computer Science,
University of Pittsburgh,
210 S Bouquet St, Pittsburgh, PA 15260, USA
{hux10,hwa}@cs.pitt.edu

Abstract

This work explores methods of automatically detecting corrections of individual mistakes in sentence revisions for ESL students. We have trained a classifier that specializes in determining whether consecutive basic-edits (word insertions, deletions, substitutions) address the same mistake. Experimental result shows that the proposed system achieves an F_1-score of 81% on correction detection and 66% for the overall system, out-performing the baseline by a large margin.

1 Introduction

Quality feedback from language tutors can help English-as-a-Second-Language (ESL) students improve their writing skills. One of the tutors' tasks is to isolate writing mistakes within sentences, and point out (1) why each case is considered a mistake, and (2) how each mistake should be corrected. Because this is time consuming, tutors often just rewrite the sentences without giving any explanations (Fregeau, 1999). Due to the effort involved in comparing revisions with the original texts, students often fail to learn from these revisions (Williams, 2003).

Computer aided language learning tools offer a solution for providing more detailed feedback. Programs can be developed to compare the student's original sentences with the tutor-revised sentences. Swanson and Yamangil (2012) have proposed a promising framework for this purpose. Their approach has two components: one to detect individual corrections within a revision, which they termed *correction detection*; another to determine what the correction fixes, which they termed *error type selection*. Although they reported a high accuracy for the error type selection classifier alone, the bottleneck of their system is the other component – correction detection. An analysis of their system shows that approximately 70% of the system's mistakes are caused by mis-detections in the first place. Their correction detection algorithm relies on a set of heuristics developed from one single data collection (the FCE corpus (Yannakoudakis et al., 2011)). When determining whether a set of basic-edits (word insertions, deletions, substitutions) contributes to the same correction, these heuristics lack the flexibility to adapt to a specific context. Furthermore, it is not clear if the heuristics will work as well for tutors trained to mark up revisions under different guidelines.

We propose to improve upon the correction detection component by training a classifier that determines which edits in a revised sentence address the same error in the original sentence. The classifier can make more accurate decisions adjusted to contexts. Because the classifier were trained on revisions where corrections are explicitly marked by English experts, it is also possible to build systems adjusted to different annotation standards.

The contributions of this paper are: (1) We show empirically that a major challenge in correction detection is to determine the number of edits that address the same error. (2) We have developed a merging model that reduces mis-detection by 1/3, leading to significant improvement in the accuracies of combined *correction detection* and *error type selection*. (3) We have conducted experiments across multiple corpora, indicating that the proposed merging model is generalizable.

2 Correction Detection

Comparing a student-written sentence with its revision, we observe that each correction can be decomposed into a set of more basic edits such as word insertions, word deletions and word substitutions. In the example shown in Figure 1, the correction "*to change* ⇒ *changing*" is composed of a deletion of *to* and a substitution from *change*

Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Short Papers), pages 599–604,
Baltimore, Maryland, USA, June 23-25 2014.

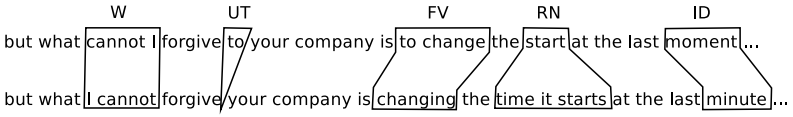


Figure 1: Detecting corrections from revisions. Our system detects individual corrections by comparing the original sentence with its revision, so that each correction addresses one error. Each polygon corresponds to one correction; the labels are codes of the error types. The codes follow the annotation standard in FCE corpus (Nicholls, 2003). In this example, *W* is incorrect Word order; *UT* is Unnecessary preposiTion; *FV* is wrong Verb Form; *RN* is Nnoun needs to be Replaced; *ID* is IDiom error.

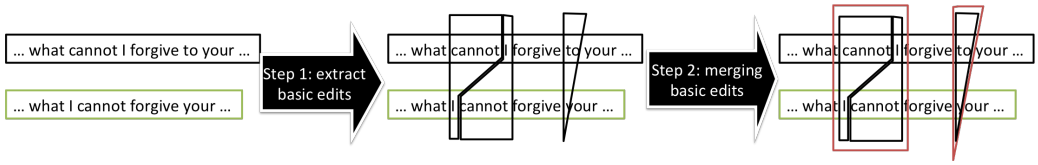
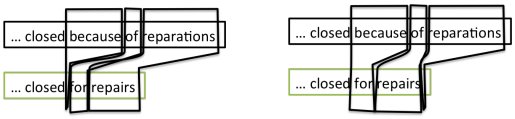
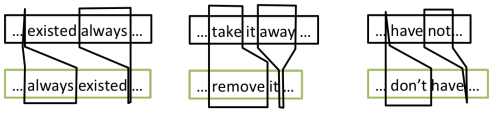


Figure 2: A portion of the example from Figure 1 undergoing the two-step correction detection process. The basic edits are indicated by black polygons. The corrections are shown in red polygons.

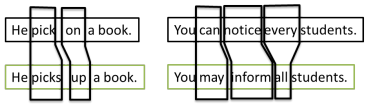


(a) (b)

Figure 3: Basic edits extracted by the edit-distance algorithm (Levenshtein, 1966) do not necessarily match our linguistic intuition. The ideal basic-edits are shown in Figure 3a, but since the algorithm only cares about minimizing the number of edits, it may end up extracting basic-edits shown in Figure 3b.



(a) The basic edits are addressing the same problem. But these basic edits are non-adjacent, and therefore not merged by S&Y's algorithm.



(b) The basic edits in the above two cases address different problems though they are adjacent. S&Y's merging algorithm incorrectly merges them.

Figure 4: Merging mistakes by the algorithm proposed in Swanson and Yamangil (2012) (S&Y), which merges adjacent basic edits.

to *changing*; the correction "*moment* ⇒ *minute*" is itself a single word substitution. Thus, we can build systems to detect corrections which operates in two steps: (1) detecting the basic edits that took place during the revision, and (2) merging those basic edits that address the same error. Figure 2 illustrates the process for a fragment of the example sentence from Figure 1.

In practice, however, this two-step approach may result in mis-detections due to ambiguities. Mis-detections may be introduced from either steps. While detecting basic edits, Figures 3 gives an example of problems that might arise. Because the Levenshtein algorithm only tries to minimize the number of edits, it does not care whether the edits make any linguistic sense. For merging basic edits, Swanson and Yamangil applied a distance heuristic – basic-edits that are close to each other (e.g. basic edits with at most one word lying in between) are merged. Figure 4 shows cases for which the heuristic results in the wrong scope.

These errors caused their system to mis-detect 30% of the corrections. Since mis-detected corrections cannot be analyzed down the pipeline, the correction detection component became the bottle-neck of their overall system. Out of the 42% corrections that are incorrectly analyzed[1], 30%/42%≈70% are caused by mis-detections in the first place. An improvement in correction detection may increase the system accuracy overall.

We conducted an error analysis to attribute errors to either step when the system detects a wrong set of corrections for a sentence. We examine the first step's output. If the resulting basic edits do not match with those that compose the actual corrections, we attribute the error to the first step. Otherwise, we attribute the error to the second step. Our analysis confirms that the merging step is the bottleneck in the current correction detection system – it accounts for 75% of the mis-detections. Therefore, to effectively reduce the algorithm's mis-detection errors, we propose to

[1]Swanson and Yamangil reported an overall system with 58% F-score.

build a classifier to merge with better accuracies.

Other previous tasks also involve comparing two sentences. Unlike evaluating grammar error correction systems (Dahlmeier and Ng, 2012), correction detection cannot refer to a gold standard. Our error analysis above also highlights our task's difference with previous work that identify corresponding phrases between two sentences, including phrase extraction (Koehn et al., 2003) and paraphrase extraction (Cohn et al., 2008). They are fundamentally different in that the granularity of the extracted phrase pairs is a major concern in our work – we need to guarantee each detected phrase pair to address exactly one writing problem. In comparison, phrase extraction systems aim to improve the end-to-end MT or paraphrasing systems. A bigger concern is to guarantee the extracted phrase pairs are indeed translations or paraphrases. Recent work therefore focuses on identifying the alignment/edits between two sentences (Snover et al., 2009; Heilman and Smith, 2010).

3 A Classifier for Merging Basic-Edits

Figures 4 highlights the problems with indiscriminantly merging basic-edits that are adjacent. Intuitively, it seems that the decision should be more context dependent. Certain patterns may indicate that two adjacent basic-edits are a part of the same correction while others may indicate that they each address a different problem. For example, in Figure 5a, when the insertion of one word is followed by the deletion of the same word, the insertion and deletion are likely addressing one single error. This is because these two edits would combine together as a word-order change. On the other hand, in Figure 5b, if one edit includes a substitution between words with the same POS's, then it is likely fixing a word choice error by itself. In this case, it should not be merged with other edits.

To predict whether two basic-edits address the same writing problem more discriminatively, we train a Maximum Entropy binary classifier based on features extracted from relevant contexts for the basic edits. We use features in Table 1 in the proposed classifier. We design the features to indicate: **(A)** whether merging the two basic-edits matches the pattern for a common correction. **(B)** whether one basic-edit addresses one single error.

We train the classifier using samples extracted from revisions where individual corrections are explicitly annotated. We first extract the basic-edits that compose each correction. We then create a training instance for each pair of two consecutive basic edits: if two consecutive basic edits need to be merged, we will mark the outcome as *True*, otherwise it is *False*. We illustrate this in Figure 6.

(a) The pattern indicates that the two edits address the same problem

(b) The pattern indicates that the the two edits do not address the same problem

Figure 5: Patterns indicating whether two edits address the same writing mistake.

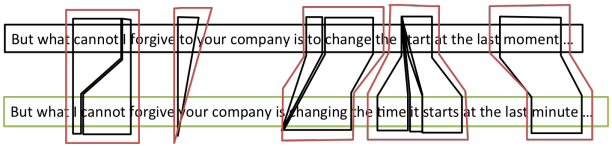


Figure 6: Extracting training instances for the merger. Our goal is to train classifiers to tell if two basic edits should be merged (*True* or *False*). We break each correction (outer polygons, also colored in red) in the training corpus into a set of basic edits (black polygons). We construct an instance for each consecutive pair of basic edits. If two basic edits were extracted from the same correction, we will mark the outcome as *True*, otherwise we will mark the outcome as *False*.

4 Experimental Setup

We combine Levenshtein algorithm with different merging algorithms for correction detection.

4.1 Dataset

An ideal data resource would be a real-world collection of student essays and their revisions (Tajiri et al., 2012). However, existing revision corpora do not have the fine-grained annotations necessary for our experimental gold standard. We instead use error annotated data, in which the corrections were provided by human experts. We simulate the revisions by applying corrections onto the original sentence. The teachers' annotations are treated as gold standard for the detailed corrections.

We considered four corpora with different ESL populations and annotation standards, including FCE corpus (Yannakoudakis et al., 2011), NUCLE corpus (Dahlmeier et al., 2013), UIUC corpus[2] (Rozovskaya and Roth, 2010) and HOO2011 corpus (Dale and Kilgarriff, 2011). These corpora all provide experts' corrections along with error

[2]UIUC corpus contains annotations of essays collected from ICLE (Granger, 2003) and CLEC (Gui and Yang, 2003).

Type	name	description
A	gap-between-edits	Gap between the two edits. In particular, we use the number of words between the two edits' original words, as well as the revised words. Note that Swanson and Yamangil's approach is a special case that only considers if the basic-edits have zero gap in both sentences.
	tense-change	We detect patterns such as: if the original-revision pair matches the pattern "V-ing⇒to V".
	word-order-error	Whether the basic-edits' original word set and the revised word set are the same (one or zero).
	same-word-set	If the original sentence and the revised sentence have the same word set, then it's likely that all the edits are fixing the word order error.
	revised-to	The phrase comprised of the two revised words.
B	editdistance=1	If one basic-edit is a substitution, and the original/revised word only has 1 edit distance, it indicates that the basic-edit is fixing a misspelling error.
	not-in-dict	If the original word does not have a valid dictionary entry, then it indicates a misspelling error.
	word-choice	If the original and the revised words have the same POS, then it is likely fixing a word choice error.
	preposition-error	Whether the original and the revised words are both prepositions.

Table 1: Features used in our proposed classifier.

corpus	sentences	$\frac{\text{sentences with} \geq 2 \text{ corrections}}{\text{revised sentences}}$
FCE	33,900	53.45%
NUCLE	61,625	48.74%
UIUC	883	61.32%
HOO2011	966	42.05%

Table 2: Basic statistics of the corpora that we consider.

Corpus	Error Types	Accuracy
FCE	73	80.02%
NUCLE	27	67.36%
UIUC	8	80.23%
HOO2011	38	64.88%

Table 3: Error type selection accuracies on different corpora. We use a Maximum Entropy classifier along with features suggested by Swanson and Yamangil for this task. The reported figures come from 10-fold cross validations on different corpora.

type mark-ups. The basic statistics of the corpora are shown in Table 2. In these corpora, around half of revised sentences contains multiple corrections. We have split each corpus into 11 equal parts. One part is used as the development dataset; the rest are used for 10-fold cross validation.

4.2 Evaluation Metrics

In addition to evaluating the merging algorithms on the stand-alone task of correction detection, we have also plugged in the merging algorithms into an end-to-end system in which every automatically detected correction is further classified into an error type. We replicated the error type selector described in Swanson and Yamangil (2012). The error type selector's accuracies are shown in Table 3[3] . We compare two merging algorithms, combined with Levenshtein algorithm:

S&Y The merging heuristic proposed by Swanson and Yamangil, which merges the adjacent basic edits into single corrections.

MaxEntMerger We use the Maximum Entropy classifier to predict whether we should merge the two edits, as described in Section 3[4].

We evaluate extrinsically the merging components' effect on overall system performance by comparing the boundaries of system's detected corrections with the gold standard. We evaluate both (1) the F-score in detecting corrections (2) the F-score in correctly detecting both the corrections' and the error types they address.

5 Experiments

We design experiments to answer two questions:

1. Do the additional contextual information about correction patterns help guide the merging decisions? How much does a classifier trained for this task improve the system's overall accuracy?

2. How well does our method generalize over revisions from different sources?

Our major experimental results are presented in Table 4 and Table 6. Table 4 compares the overall educational system's accuracies with different merging algorithms. Table 6 shows the system's F_1 score when trained and tested on different corpora. We make the following observations:

First, Table 4 shows that by incorporating correction patterns into the merging algorithm, the

[3]Our replication has a slightly lower error type selection accuracy on FCE (80.02%) than the figure reported by Swanson and Yamangil (82.5%). This small difference on error type selection does not affect our conclusions about correction detection.

[4]We use the implementation at `http://homepages.inf.ed.ac.uk/lzhang10/maxent_toolkit.html`.

errors in correction detection step were reduced. This led to a significant improvement on the overall system's F_1-score on all corpora. The improvement is most noticeable on FCE corpus, where the error in correction detection step was reduced by 9%. That is, one third of the correction mis-detections were eliminated. Table 5 shows that the number of merging errors are significantly reduced by the new merging algorithm. In particular, the number of false positives (system proposes merges when it should not) is significantly reduced.

Second, our proposed model is able to generalize over different corpora. As shown in Table 6. The models built on corpora can generally improve the correction detection accuracy[5]. Models built on the same corpus generally perform the best. Also, as suggested by the experimental result, among the four corpora, FCE corpus is a comparably good resource for training correction detection models with our current feature set. One reason is that FCE corpus has many more training instances, which benefits model training. We tried varying the training dataset size, and test it on different corpora. Figure 7 suggests that the model's accuracies increase with the training corpus size.

6 Conclusions

A revision often contains multiple corrections that address different writing mistakes. We explore building computer programs to accurately detect individual corrections in one single revision. One major challenge lies in determining whether consecutive basic-edits address the same mistake. We propose a classifier specialized in this task. Our experiments suggest that: (1) the proposed classifier reduces correction mis-detections in previous systems by 1/3, leading to significant overall system performance. (2) our method is generalizable over different data collections.

Acknowledgements

This work is supported by U.S. National Science Foundation Grant IIS-0745914. We thank the anonymous reviewers for their suggestions; we also thank Homa Hashemi, Wencan Luo, Fan Zhang, Lingjia Deng, Wenting Xiong and Yafei Wei for helpful discussions.

[5]We currently do not evaluate the end-to-end system over different corpora. This is because different corpora employ different error type categorization standards.

Method	Corpus	Correction Detection F_1	Overall F_1-score
S&Y	FCE	70.40%	57.10%
MaxEntMerger	FCE	**80.96%**	**66.36%**
S&Y	NUCLE	61.18%	39.32%
MaxEntMerger	NUCLE	**63.88%**	**41.00%**
S&Y	UIUC	76.57%	65.08%
MaxEntMerger	UIUC	**82.81%**	**70.55%**
S&Y	HOO2011	68.73%	50.95%
MaxEntMerger	HOO2011	**75.71%**	**56.14%**

Table 4: Extrinsic evaluation, where we plugged the two merging models into an end-to-end feedback detection system by Swanson and Yamangil.

Merging algorithm	TP	FP	FN	TN
S&Y	33.73%	13.46%	5.71%	47.10%
MaxEntMerger	36.04%	3.26%	3.41%	57.30%

Table 5: Intrinsic evaluation, where we evaluate the proposed merging model's prediction accuracy on FCE corpus. This table shows a breakdown of true-positives (TP), false-positives (FP), false-negatives (FN) and true-negatives (TN) for the system built on FCE corpus.

training \ testing	FCE	NUCLE	UIUC	HOO2011
S&Y	70.44	61.18%	76.57%	68.73%
FCE	**80.96%**	61.26%	**83.07%**	75.43%
NUCLE	74.53%	**63.88%**	78.57%	74.73%
UIUC	77.25%	58.21%	82.81%	70.83%
HOO2011	71.94%	54.99%	71.19%	**75.71%**

Table 6: Correction detection experiments by building the model on one corpus, and applying it onto another. We evaluate the correction detection performance with F_1 score. When training and testing on the same corpus, we run a 10-fold cross validation.

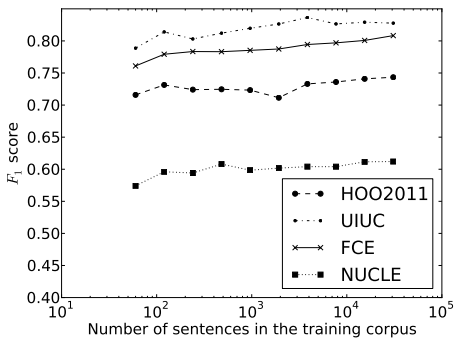


Figure 7: We illustrate the performance of correction detection systems trained on subsets of FCE corpus. Each curve in this figure represents the F_1-scores for correction detection of the model trained on a subset of FCE and tested on different corpora. When testing on FCE, we used $\frac{1}{11}$ of the FCE corpus, which we kept as development data.

References

Trevor Cohn, Chris Callison-Burch, and Mirella Lapata. 2008. Constructing corpora for the development and evaluation of paraphrase systems. *Computational Linguistics*, 34(4):597–614.

Daniel Dahlmeier and Hwee Tou Ng. 2012. Better evaluation for grammatical error correction. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 568–572, Montréal, Canada, June. Association for Computational Linguistics.

Daniel Dahlmeier, Hwee Tou Ng, and Siew Mei Wu. 2013. Building a large annotated corpus of learner english: The NUS corpus of learner english. In *Proceedings of the Eighth Workshop on Innovative Use of NLP for Building Educational Applications*, pages 22–31.

Robert Dale and Adam Kilgarriff. 2011. Helping our own: The HOO 2011 pilot shared task. In *Proceedings of the 13th European Workshop on Natural Language Generation*, pages 242–249. Association for Computational Linguistics.

Laureen A Fregeau. 1999. Preparing ESL students for college writing: Two case studies. *The Internet TESL Journal*, 5(10).

Sylviane Granger. 2003. The International Corpus of Learner English: a new resource for foreign language learning and teaching and second language acquisition research. *Tesol Quarterly*, 37(3):538–546.

Shicun Gui and Huizhong Yang. 2003. Zhongguo xuexizhe yingyu yuliaohu.(chinese learner english corpus). *Shanghai: Shanghai Waiyu Jiaoyu Chubanshe*.

Michael Heilman and Noah A Smith. 2010. Tree edit models for recognizing textual entailments, paraphrases, and answers to questions. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 1011–1019. Association for Computational Linguistics.

P. Koehn, F.J. Och, and D. Marcu. 2003. Statistical phrase-based translation. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, pages 48–54. Association for Computational Linguistics.

V. I. Levenshtein. 1966. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10(8):707710.

D. Nicholls. 2003. The Cambridge Learner Corpus: Error coding and analysis for lexicography and ELT. In *Proceedings of the Corpus Linguistics 2003 conference*, pages 572–581.

Alla Rozovskaya and Dan Roth. 2010. Annotating ESL errors: Challenges and rewards. In *Proceedings of the NAACL HLT 2010 fifth workshop on innovative use of NLP for building educational applications*, pages 28–36. Association for Computational Linguistics.

Matthew G Snover, Nitin Madnani, Bonnie Dorr, and Richard Schwartz. 2009. TER-Plus: paraphrase, semantic, and alignment enhancements to translation edit rate. *Machine Translation*, 23(2-3):117–127.

Ben Swanson and Elif Yamangil. 2012. Correction detection and error type selection as an ESL educational aid. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 357–361, Montréal, Canada, June. Association for Computational Linguistics.

Toshikazu Tajiri, Mamoru Komachi, and Yuji Matsumoto. 2012. Tense and aspect error correction for ESL learners using global context. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Short Papers-Volume 2*, pages 198–202. Association for Computational Linguistics.

Jason Gordon Williams. 2003. Providing feedback on ESL students written assignments. *The Internet TESL Journal*, 4(10).

Helen Yannakoudakis, Ted Briscoe, and Ben Medlock. 2011. A new dataset and method for automatically grading esol texts. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 180–189. Association for Computational Linguistics.