Assisting teachers and annotators in their tasks: semi-automatic ensemble-based correction of errors

Anonymous ACL submission

Abstract

We present in this paper an experiment to support the work of correcting errors in learner texts. Our system suggests possible corrections for spans of text that have been manually marked as erroneous, such as when teachers correct learner essays or when linguists annotate errors in a learner corpus. Our system uses a two-step ensemble approach relying first on several components capable of suggesting candidate corrections, most of them being specifically trained machine translation (MT) systems, and then on a random forest machine learning (ML) classifier that categorizes candidate corrections as correct or incorrect. The results we obtained on German data show that our approach has tangible potential, but also that a number of challenges remain to be addressed in order to truly exploit it in practice.

1 Introduction

Manually correcting errors is a task that can be time-consuming, tedious, and error-prone. The experiment and results we present in this paper focus on the semi-automatic correction of errors that have been identified and marked. Such support aims both at reducing the time needed for tasks such as the correction of learner essays by teachers or the annotation of errors in learner corpora used to perform linguistic studies, and at increasing the overall quality and homogeneity of corrections by providing experts with an automatic tool that assists them in a systematic fashion.

In a two-step approach, we start by training several Statistical Machine Translation (SMT) systems to translate learner language into standard language and generate monolingual candidate corrections for erroneous spans of texts. We also run the Hunspell spell-checker[1] to generate additional candidate corrections. In a second step, we combine the heterogeneously overlapping sets of candidate corrections generated in an ensemble fashion (Opitz and Maclin, 1999) and provide them all as an input to a random forest classifier (Ho, 1995) that we train to categorize the candidate corrections as valid or invalid and rank them according to the confidence score provided by the classifier.

This work complements previous efforts such as the ones by Okinina and Nicolas (2018) that demonstrated that even standard, non-thoroughly tuned SMT models based on (Koehn et al., 2007) and trained with the default configuration can achieve promising results when considering orthographic errors. For this work, we further explored this question by adding an ensemble component that dynamically combines the outputs of several systems capable of generating candidate corrections.

This paper is organized as follows. In Section 2, we discuss the state of the art and its relation to our work. In section 3, we describe our targeted use-case and the approach itself. We also explain how our experiment was shaped to test the approach and present its setting, be it in terms of training and testing data or in terms of the systems that we trained and tested on the data. We then detail in Section 4 the results from the experiment and discuss in Section 5 how we intend to address the observed shortcomings and extend the approach. Finally, we conclude in Section 6.

2 Related Work

The state of the art (SOTA) contains several methods relying on SMT and Neural Machine Translation (NMT) (Chollampatt and Ng, 2017; Ji et al., 2017; Junczys-Dowmunt and Grundkiewicz, 2016; Napoles et al., 2017; Sakaguchi et al., 2017; Schmaltz et al., 2017; Yuan and Briscoe, 2016; Grundkiewicz et al., 2019; Choe et al., 2019) and tackle a task referred to as grammatical error correction (GEC). Similarly to our first step, these approaches treat error correction as an MT task and translate content from incorrect to correct language and, in our case, from learner language to

[1] github.com/hunspell/hunspell

standard language. When it comes to orthographic error correction, these “languages” are extremely close, which greatly facilitates the MT task, similarly to the task of translating closely-related languages such as Macedonian and Bulgarian (Nakov and Tiedemann, 2012). Some works approach the problem as sequence tagging, for example, like Omelianchuk et al. (2020) who designs custom token-level transformations to map input tokens to target corrections.

The SOTA also includes related works demonstrating that, for both MT and GEC purposes, an ensemble approach can achieve noticeable improvements over the single systems it relies on (Dabre et al., 2019; Garmash and Monz, 2016; Choe et al., 2019; Náplava and Straka, 2019; Katsumata and Komachi, 2020). Besides Okinina and Nicolas (2018), most of the existing approaches automatically process whole sentences and can thus introduce unnecessary incorrect corrections on error-free spans of text. Some previous works (Chen et al., 2020) divide the task of GEC into two sub-tasks: erroneous span detection and erroneous span correction. Like Okinina and Nicolas (2018), we follow this distinction but leave the detection of the erroneous spans of the text to the user. This pragmatic choice is meant to "foster" user adoption as it limits the generation of false-positive corrections, which can be time-consuming and discouraging for our targeted users (i.e., teachers and annotators). Indeed, as far as our understanding on the question goes, error detection is a challenge that is not yet addressed with a quality that would satisfy our end-users (e.g. in Bell et al., 2019, the highest precision for grammatical error detection in English is 79.51). We thus decided to focus our efforts, for the time being, on a sub-task for which we can achieve better results in terms of precision, and could eventually allow us in the future to devise a new type of approach to detect errors in a brute-force backward fashion (see related discussion in Section 5). To the best of our knowledge, our work is the only one besides (Okinina and Nicolas, 2018) that has approached the question in such a manner, as all the previous related works aiming at correcting texts, that we are aware of, also perform some kind of error detection.

3 Approach and Experimental Setting

In this section, we discuss the use-cases targeted by our approach (Section 3.1), then present the two steps it is composed of (Section 3.2). We finally provide details about the settings of our experiment and how the method was implemented and tested in practice (Section 3.3).

3.1 Targeted Use Cases

Unlike standard language, which is normalized, learner language can greatly vary within a group of learners. In other words, the errors made by a certain group of learners when learning a second language (i.e., beginner English learners with French L1) can greatly differ from the errors made by another group of learners learning the same language (e.g., advanced English learners with Spanish L1). This phenomena is particularly observable among learners with different backgrounds (e.g., different L1-s) or levels of proficiency (e.g., beginner vs advanced learners) (Jarvis and Pavlenko, 2008).

Consequently, training a correction system on errors made by one group of learners and using it to correct the errors made by another group of learner will inherently lead to poorer results. Overall, this issue is not limited to our use cases and applies to all systems for which training data differ significantly from the data they are used on. However, the results achieved by Okinina and Nicolas (2018), as well as the ones we obtained while exploring this question, have shown that this point is particularly relevant for our intended use case.

Therefore, for the use-case we tackle, both the data used to train the systems we use and the data on which they are used should be highly coherent with one another. In other words, both datasets should ideally be two sub-parts of a larger dataset. For a teacher correcting essays, this means using the essays previously corrected over time for a specific school and a specific part of the curricula as training data in order to support the correction of new essays. For a linguist annotating errors in a learner corpus, this means using an already annotated part of the corpus as training data in order to support the annotation of the remaining part.

In both cases, as the amount of data could be limited at first, the performance of data-intensive tools such as MT systems will greatly vary. Implementing our approach into a proper tool will thus imply finding viable strategies to regularly re-train the tools over multiple iterations every time the training dataset has sufficiently increased in size. As such, the experiment we describe in this paper corresponds to a single one of these foreseen itera-

tions, one with a favourable setting allowing us to confirm the potential of the approach, whereas the challenging question of how to best implement our method in practice is left for future works.

3.2 A Two-step Approach

In a first step, we trained and tested SMT systems with 10-fold cross-validation on a bilingual learner-language/standard-language corpus. When doing so, the SMT systems were constrained to only correct the spans of text that were marked as containing an error. We recorded the five best candidate corrections they generated, as well as the scores associated with them by the MT systems. We also ran a spell-checker to get a maximum of five suggested corrections. We hereafter refer to the SMT systems and the spell checker we used as *correction guessers*.

In a second step, we trained and tested with 10-fold cross-validation an ML-based classifier on a randomized corpus composed of the heterogeneously overlapping sets of candidate corrections generated during the first step when running the correction guessers on test sets. As input to the classifier, we do not provide the actual candidate correction but represent it as the following set of features for each correction guesser: a unique identifier for all suggestions pertaining to the same error span, a Boolean indicating whether a specific guesser suggested the given correction, the score a guesser assigned to a correction, and the rank within the sets of generated candidate corrections. In addition, because the score from a guesser, for example from Moses, can be unbound, we add features where we scale, standardize, and normalize[2] the score within each unique identifier; and we also standardize and normalize the rank.

Each candidate correction is also provided with a Boolean feature indicating if it is equal to the "gold" correction associated with the error in the learner corpus[3]. We thus train the ML-based classifier to recognize valid candidate corrections by balancing, for each candidate correction, the "opinions" of the correction guessers (or absence of opinions if a correction guesser has not generated it). In practice, the ML-based classifier associates with each candidate correction a category (valid vs invalid) and a score that we use to rank and select the candidate corrections categorized as valid.

3.3 Experimental Setting

This experiment relied on three corpus, two of them being automatically generated.

The first learner corpus, the Koko corpus[4] (Abel et al., 2016) licensed under ACA-BY-NC-NORED 1.0, is used to generate a "bilingual" learner-language/standard-language parallel corpus by simply duplicating each sentences twice: the original sentence written by the learners where each error is annotated and the same sentence where errors are corrected. The Koko corpus is composed of around 1503 written learner essays (around 900k tokens) written by secondary-school pupils, learning German as their mother tongue. Beside a gold correction, each error annotation is associated with an error type.

The second corpus, the bilingual learner-language/standard-language corpus that we generated automatically from the Koko corpus, has the following properties: it is composed of 8.5 million characters, 1.8 million tokens, and 49 186 pairs of bilingual sentences of which 11 307 contain one or several of the 15 039 errors. The errors themselves contain 127k characters and 19k tokens. Overall, this bilingual corpus is of rather moderate size when compared to bilingual corpora of several million tokens that MT systems are usually trained on. This corpus is used during the first step to train the SMT-based correction guessers and generate sets of candidate corrections.

Finally, the third corpus is composed of partially overlapping sets of candidate corrections and contains 61 500 candidate corrections, including 8 968 valid candidate corrections (14.6% of all candidate corrections) split over 9 244 sets of candidate corrections. Therefore, 9 244 errors (61.1% of the 15 039 errors) have received a set of one or more candidate corrections whereas the correction guessers did not suggest any candidate corrections for the remaining 5 795 errors. These candidate corrections are used in the second step to train and test the ML-based classifier.

3.4 SMT Systems

For our experiment, we relied on the implementation of SMT models provided by the version 4.0

[2] We use `preprocessing.normalize`, `StandardScaler`, `MaxAbsScaler` from scikit-learn (Pedregosa et al., 2011).

[3] Even though there can be several valid corrections for a given error, we currently use that information to decide if a candidate correction is valid or not.

[4] https://www.porta.eurac.edu/lci/koko/

of the Moses toolkit (Koehn et al., 2007) licensed under the LGP and made available on the official website[5]. This choice was driven by three reasons. First, our ensemble approach could accommodate many types of correction guessers. As such, relying on SMT does not prevent us from including other types of correction guessers at a later stage, including guessers relying on other MT technologies. Second, the aim of this experiment was to explore the validity of the approach. Therefore, using a well-established toolkit was a safe strategy to achieve reliable results. Third, SMT is less data-intensive than other MT technologies, so it seemed a better fit for our small parallel corpus.

For our first experiment, we used the default configuration of the Moses toolkit and trained four different SMT models relying either on word or character n-grams (a 1-gram, a 3-gram, and 5-gram word model and a 10-gram character model). By doing so, we aimed at creating different correction guessers capable of addressing different sets of errors. Indeed, since some errors are hardly related to the context or tokens in which they appear (e.g., spelling errors), they can be addressed by a character-based and a token-independent model. At the same time, some errors are related to their context (e.g., agreement errors) and can not be addressed without contextual information. The reason for using n-gram models of different size is also motivated by the limited size of our parallel corpus: the larger the considered n-grams the more sparse this limited set of data will be. We thus trained several types of SMT models with the intuition that they will behave differently [6].

Since the SMTs did not always respect the restrictions set on the part of the text to correct, we could not always align the corrected sentence to the one provided as an input and extract the candidate corrections they contained. Accordingly, 4 847 errors stayed without any candidate, even though the SMT models did output corrected versions of sentences containing these errors.

3.5 Spell-checker

Hunspell is a free spell checker and morphological analyzer library and command-line tool, licensed under LGPL/GPL/MPL tri-license. We used the version 1.7.0 that is made available on a dedicated github repository[7]. Hunspell checks if a word is correct (lookup) and, for a misspelled word, provides a hypothesis how it might be corrected (suggest). In order to provide a suggestion, Hunspell computes the similarity of the misspelled word to all words in its dictionary and chose the most similar ones. Because Hunspell does not provide a confidence score with the suggested corrections, we computed a Levenshtein distance between the candidate correction and the text of the errors.

3.6 ML Classifier

For the ML-based classifier used during the second step, we relied on the implementation of the random forest algorithm offered by the version 3.8 of the Weka toolkit[8] (Frank et al., 2016), licensed under the GNU General Public license 3.0, with the default configuration[9] provided.

4 Results

4.1 Training the SMT models

Since we aimed at performing a preliminary evaluation of the approach, we devised this experiment with a favorable setting in which the SMT-based correction guessers are trained for the first step on 90% of the bilingual learner-language/standard-language parallel corpus and tested on the remaining 10%. We thus performed a 10-fold procedure and, by doing so, simulated a situation in which the supported expert has 9 times more corrected errors than errors to correct.

In order to ensure that the 10-fold cross-validation procedure was properly implemented, we computed statistics regarding the error annotations over the different folds. Indeed, by simply checking how much the ratio of error types over the folds matched the ratio of error types found in the whole corpus, we were able to indirectly observe if the folds had an undesired bias. Table 1 below details the ratio of the 9 most frequent types of errors present in the bilingual corpus, plus an overall "other" category accounting for the remaining types of errors. It also details the average ratios of error types observed and the average statistical deviation from the ratios observed over the 10-folds for both the training and testing datasets. As no unde-

[5] https://www.statmt.org/moses/

[6] The choice of the types of n-grams used was based on preliminary tests. As the results we obtained already allowed us to meet our objective of proving the potential of our approach, we didn't try with other types of n-grams.

[7] https://github.com/hunspell/hunspell

[8] https://www.cs.waikato.ac.nz/ml/weka/

[9] `weka.classifiers.trees.RandomForest -P 100 -I 100 -K 0 -M 1.0 -V 0.001 -S 1`

sired bias was observable, we have good reasons to believe that the 10-fold procedure was properly implemented.

4.2 Generating candidate corrections

For the SMT-based correction guessers, 10 sets of candidate corrections were generated while testing the ten versions of the trained models (one during each fold) and aggregated into a single corpus of sets of candidate corrections[10]. Each error has been assigned to a single fold and used nine times for training and once for testing. By doing so, we associated with each error a set of candidate corrections whenever the SMT-based models did manage to generate some. Regarding the correction guesser relying on Hunspell, we simply ran the tool on the errors that received candidate corrections from the MT-based guessers because the performance of the spell checker was rather limited (see Table 2). Accordingly, the sets of candidate corrections it generated were not of sufficient quality to be considered on their own. Overall, no correction guessers perform satisfyingly on their own in terms of precision[11] and recall[12], while the ones based on the 3-gram and 5-gram word models tend to perform better than the others.

The correlation between the five correction guessers can be observed in Table 3. The similarities between them follow the expected behavior: the spell checker and the 10-gram-character SMT-based model are mostly dis-correlated from the other guessers while the three word-based guessers are correlated to some extent, especially the ones relying on the 3-gram and 5-gram models. Such a heterogeneous spectrum of correlations validates the strategy of creating multiple complementary SMT models in order to balance contextual window size and data scarcity.

4.3 Classifying and ranking corrections

The sets of candidate corrections generated by the correction guessers were randomized and divided in ten folds to train and test the random forest classifier trained to select valid candidates. We once more double-checked that this 10-fold procedure was properly implemented by studying if any undesired bias in terms of error types was observable over the different folds[13] (see Table 4) and, in the absence of such bias, concluded that the procedure was correctly implemented.

Table 5 summarizes the performance of the random forest component in categorizing candidate corrections as valid or invalid when using all the 61 500 candidate corrections. Candidate corrections labelled as valid are indeed valid for 96.4% of them (8 281 out of 8 587) while the ones labelled as invalid are indeed invalid for 98.7% of them (52 226 out of 52 913). Overall, such component allows us to filter most invalid candidates at the cost of loosing some correct ones. In other words, it boosts precision while reducing recall.

4.4 Ensemble Combinations

Table 6 displays the performance of the five correction guessers alone, the performance of different ensemble combinations including the combination relying on the candidate corrections generated by all five correction guessers, which we refer to hereafter as *ensemble-all*[14].

As one can observe, all ensemble combinations outperform the single correction guessers in terms of precision but not in terms of recall. As expected, the pipelines relying on single guessers have higher precision but lower recall since the random forest classifier acts as a filter. Nonetheless, combining more guessers allows to noticeably regain recall while further improving precision. The highest precision, recall and f-score achieved by one correction guesser were *93.39%*, *51.39%* and *66.23%* while the ones achieved by one of the pipelines combining several guessers and the random forest classifier were *97.26%*, *55.04%* and *70.25%*. respectively. The gain in terms of precision is particularly interesting (-60.1% invalid correction suggested) given the expectations of the users we target (see Section 3.1), whereas the gain in terms of recall remains limited (+.7.1% valid correction suggested).

[10] This implies that we consider all ten versions of the SMT models to be equivalent, which is only partially true as their training corpora each had 88% overlap with the other nine training corpora. Thus, we assumed and observed to some extent that all 10 instances of a same SMT model behave similarly without formally confirming it.

[11] Number of valid candidate corrections vs number of candidate corrections.

[12] Number of valid candidate corrections suggested vs number of errors.

[13] The list of error types in the table differs from the list presented in Table 4 since the correction guessers had varying performance with respect to error types and, accordingly, their representation in that step changed.

[14] The precision observable in for this combination (97.2%, see Table 6) is higher than the one reported for the random tree forest component (96.4%, see Table 5) since we used the confidence scores to decide on a final candidate correction when multiples ones were categorized as valid.

Error type	Corpora	Training		Testing	
		Ratio	Dev	Ratio	Dev
['orth: 02 lcp instead of cap: other cases']	15.87%	15.87%	0.18%	15.86%	1.61%
['orth: 03 cap instead of lcp']	11.10%	11.11%	0.13%	11.16%	1.22%
['orth: 11 omissions: of double consonants']	9.37%	9.37%	0.08%	9.36%	0.67%
['orth: 06 sep instead of tog: compounds']	7.67%	7.67%	0.09%	7.67%	0.81%
['orth: 14 omissions: of one grapheme']	4.80%	4.80%	0.08%	4.78%	0.75%
['corr: case']	4.16%	4.16%	0.08%	4.13%	0.74%
['orth: 07 sep instead of tog: other cases']	3.96%	3.96%	0.05%	3.96%	0.48%
['orth: 28 Eigennamen']	3.48%	3.48%	0.09%	3.50%	0.81%
['orth: 15 insertions: of double consonants']	3.05%	3.05%	0.06%	3.05%	0.51%
other	36.54%	36.54%	0.15%	36.52%	1.43%

Table 1: Ratio of error types in the corpus, training and testing datasets used for the SMT models.

ErrorType	5-gram (word)	3-gram (wrod)	1-gram (word)	10-gram (char)	spellchecker
['orth: 02 lcp instead of cap: other cases']	98.4/61.8/75.9	99.1/62.0/76.3	94.8/56.1/70.5	32.8/7.4/12.1	82.4/10.8/19.1
['orth: 03 cap instead of lcp']	95.0/57.8/71.9	95.2/58.0/72.1	89.3/52.1/65.8	81.9/33.0/47.0	3.4/0.1/0.1
['orth: 11 omissions: of double consonants']	99.1/75.6/85.7	99.3/75.9/86.0	89.1/65.2/75.3	46.2/20.9/28.8	65.7/6.5/11.9
['orth: 06 sep instead of tog: compounds']	91.5/48.6/63.5	91.5/48.5/63.4	1.1/0.5/0.7	93.3/25.5/40.1	0.0/0.0/0.0
['orth: 14 omissions: of one grapheme']	96.1/37.1/53.5	96.4/37.3/53.7	92.1/33.8/49.4	54.6/14.0/22.3	58.3/20.4/30.2
['corr: case']	68.8/34.9/46.3	71.4/36.3/48.1	71.6/34.2/46.3	1.9/0.5/0.8	0.0/0.0/0.0
['orth: 07 sep instead of tog: other cases']	85.0/59.1/69.7	86.2/59.7/70.6	2.0/1.3/1.6	80.4/26.8/40.3	0.0/0.0/0.0
['orth: 28 Eigennamen']	99.5/84.3/91.3	99.8/84.9/91.7	97.8/75.7/85.3	94.0/53.7/68.4	92.1/75.9/83.2
['orth: 15 insertions: of double consonants']	98.0/62.7/76.5	97.6/62.5/76.2	93.6/57.7/71.4	65.3/30.3/41.4	77.5/28.5/41.7
other	87.4/37.5/52.5	88.0/37.7/52.7	70.5/24.2/36.0	72.1/23.5/35.5	65.3/18.0/28.3

Table 2: Precision, recall, and F1-score of the correction guessers according to the types of errors.

Overall, the *ensemble-all* achieved slightly higher results than the five pipelines relying on the combination of four guessers. Adding the least correlated guessers (the ones relying either on the spellchecker and the 10-gram character model, see Table 3) to the combination of the other three guessers allows obtaining noticeable gain in performance. It confirms that the less a guesser is correlated with other guessers, the more it might contribute to a combination of them (which is an expected behaviour for an ensemble approach).

As detailed in Table 7, for all error types, the *ensemble-all* pipeline achieves higher precision than any of the guessers as well as higher recall and F-score for all error types except one. This result confirms that adding more correction guessers allows to boost performance, without any major backfiring on a subset of error types.

Last but not least, the two curves shown in Figure 1 represent the precision and recall of *ensemble-all* according to a minimal confidence score threshold (e.g., all corrections generated with a lower score are discarded) for the candidate corrections predicted as valid by the random forest component. As expected, the higher the threshold is set, the higher the precision becomes while the recall decreases. Such a controllable behaviour opens the possibility to allow our target users to trade precision for recall. For example, for the ensemble-all combination, considering only the 96.8% candidate corrections predicted as valid with the highest confidence scores allowed us to further enhance the precision to 98%.

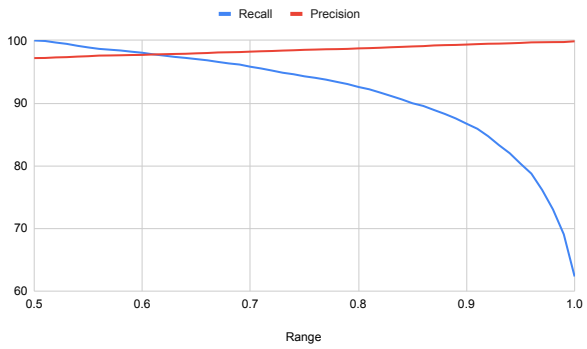

Figure 1: Confidence score vs Precision and Recall for candidate corrections predicted as correct.

5 Future Work

Our efforts and results could be extended in four directions: overcoming technical challenges, carrying out additional evaluation, improving performance, or extending the approach itself.

	10-gram (char)	1-gram (word)	3-gram (word)	5-gram (word)	Spell-checker
10-gram-character	1.0	-0.24	-0.20	-0.20	0.32
1-gram-word	-0.24	1.0	0.27	0.26	0.01
3-gram-word	-0.20	0.27	1.0	0.73	-0.01
5-gram-word	-0.20	0.26	0.73	1.0	-0.01
Spell-checker	0.32	0.01	-0.01	-0.01	1.0

Table 3: Correlation between correction guessers.

Error type	All	Training		Testing	
		Ratio	Dev	Ratio	Dev
['orth: 11 omissions: of double consonants']	16.40%	16.40%	0.16%	16.39%	1.46%
['orth: 02 lcp instead of cap: other cases']	12.35%	12.35%	0.13%	12.35%	1.20%
['orth: 06 sep instead of tog: compounds']	9.87%	9.87%	0.14%	9.87%	1.24%
['orth: 03 cap instead of lcp']	8.58%	8.58%	0.07%	8.58%	0.64%
['orth: 07 sep instead of tog: other cases']	6.74%	6.74%	0.08%	6.73%	0.75%
['orth: 28 Eigennamen']	4.69%	4.69%	0.07%	4.69%	0.60%
['orth: 08 tog instead of sep: minimal phraseologism']	4.11%	4.11%	0.07%	4.11%	0.59%
['corr: case']	4.00%	4.00%	0.06%	3.99%	0.56%
['orth: 15 insertions: of double consonants']	3.98%	3.98%	0.06%	3.98%	0.52%
other	29.29%	29.29%	0.20%	29.30%	1.80%

Table 4: Ratio of error types in the sets of candidate corrections used to train and test the random forest.

Actual/Predicted	valid	invalid	Accuracy
valid	52226	306	99.42%
invalid	687	8281	92.34%
Accuracy	98.70%	96.44%	

Table 5: Performance of the random forest component in categorizing candidate corrections.

System	Precision	Recall	F-score
Pipeline with a single Correction guessers on its own			
spellcheker	70.70%	13.41%	22.55%
10-gram (char)	65.76%	21.91%	32.87%
1-gram (word)	74.34%	37.16%	49.55%
3-gram (word)	93.12%	51.39%	66.23%
5-gram (word)	92.62%	51.14%	65.90%
Pipeline with ensemble combinations			
Ensemble_{sc}	90.47%	13.63%	23.69%
Ensemble_{10c}	90.58%	18.87%	31.23%
Ensemble_{1w}	93.21%	35.32%	51.23%
Ensemble_{3w}	96.16%	50.42%	66.15%
Ensemble_{5w}	96.07%	50.26%	65.99%
$\text{Ensemble}_{sc/10c}$	92.97%	24.18%	38.37%
$\text{Ensemble}_{1w/3w/5w}$	96.50%	50.53%	66.33%
$\text{Ensemble}_{sc/10c/1w/3w}$	96.91%	**55.04%**	70.20%
$\text{Ensemble}_{sc/10c/1w/5w}$	96.98%	54.90%	70.11%
$\text{Ensemble}_{sc/10c/3w/5w}$	97.05%	54.90%	70.13%
$\text{Ensemble}_{sc/1w/3w/5w}$	96.09%	52.16%	67.61%
$\text{Ensemble}_{10c/1w/3w/5w}$	**97.26%**	54.31%	69.70%
Ensemble-all	97.19%	55.00%	**70.25%**

Table 6: Performance of the different pipelines.

5.1 Technical Challenges

As our method could require several iterations of training, the approach could become too computationally intensive for integration into a seamless workflow on regular computers. We even faced this issue during our (limited) experiment, as we had restrained the number of word-based and character-based SMT models because of a lack of computational power to train them. We will address this challenge by exploring computationally efficient technologies while remotely delegating some computations to computing grid.

Relying on a toolkit which code is growing outdated is incompatible with our wish to create a sustainable tool. As such, the obsolescence of Moses is also a technical challenge since, as time passes and its user base naturally diminishes, properly maintaining and operating it becomes a greater challenge.We will thus have to fully replace it with another toolkit or stop using SMT models.

5.2 Additional Evaluation

The Koko corpus is mostly error-annotated for orthographic errors. It would be interesting to confront our approach with other types of error in order to double-check how much generic it is.

Since the training data we use is rather limited, performance could probably be enhanced by "simply" accumulating more training data. By divid-

Error type	ensemble-all			Best competitor			
	Precision	Recall	F-score	Label	Precision	Recall	f-score
['orth: 02 lcp instead of cap: other cases']	**99.41%**	**63.37%**	**77.40%**	3-gram	99.1%	62.0%	76.3%
['orth: 03 cap instead of lcp']	**98.88%**	**63.59%**	**77.41%**	3-gram	95.2%	58.0%	72.1%
['orth: 11 omissions: of double consonants']	**99.46%**	**78.07%**	**87.48%**	3-gram	99.3%	75.9%	86.0%
['orth: 06 sep instead of tog: compounds']	**99.00%**	**51.60%**	**67.84%**	5-gram	91.5%	48.6%	63.5%
['orth: 14 omissions: of one grapheme']	**99.05%**	**43.21%**	**60.17%**	3-gram	96.4%	37.3%	53.7%
['corr: case']	**78.69%**	30.72%	44.19%	3-gram	71.4%	**36.3%**	**48.1%**
['orth: 07 sep instead of tog: other cases']	**97.23%**	**64.77%**	**77.74%**	3-gram	86.2%	59.7%	70.6%
['orth: 28 Eigennamen']	**99.78%**	**86.23%**	**92.51%**	3-gram	98.9%	84.1%	90.9%
['orth: 15 insertions: of double consonants']	**100%**	**71.46%**	**83.35%**	5-gram	98.0%	62.7%	76.5%
other	**94.38%**	**42.47%**	**58.58%**	3-gram	88.0%	37.7%	52.7%

Table 7: Performance per error type of the *ensemble-all* combination vs the best performing guesser.

ing our training corpus in subcorpora of different smaller sizes, we could observe the evolution of the performance and extrapolate the gain we would obtain by increasing our training dataset.

5.3 Improving Performance

As our approach can accommodate multiple correction guessers, the most direct manner to enhance performance (especially in terms of recall), would be to include more correction guessers. Accordingly, we would favour guessers relying on different technologies or approaches, such as Neural Machine Translation (NMT) models for GEC.

Taking advantage of token-related information (e.g., POS and lemma) of the tokens within the erroneous spans or within the context next to the erroneous spans could also be an interesting path to improve performance. Indeed, since some errors are token-dependent (e.g., misspelling) while other are context-dependent (e.g., gender agreement), such information could be an interesting feature to provide to the random forest classifier. Also, associating a potential correction with an error label could further help deciding if a candidate correction is correct or not (see discussion after).

It would certainly also be interesting to test other technologies to categorize and rank candidate corrections. In that perspective, it would be worth testing if recent approaches based on neural networks could be applied to our targeted use case.

5.4 Extending the method

After correcting an error, annotators usually associate errors with labels. We could observe that some types of errors are frequently occurring in specific words and/or within specific contexts and/or require a generic correction (e.g., adding a generic suffix at the end of a word). Thus, we could explore the possibility to automatically associate candidate correction with a candidate error label. If such labels can be assigned with sufficient quality, they could become another piece of useful information to use in the second step to decide on the validity of a candidate correction.

Because the precision of the random tree forest is satisfying, and especially when ruling out invalid candidate corrections, we could explore the possibility to detect errors in a (somehow backward) brute-force fashion in which we start by allowing correction guessers to change any spans of text in a sentence. We would then consider these spans as if they had been manually marked by a linguist and proceed as usual. By selecting a candidate correction, our approach would also validate the spans of text as containing errors.

6 Conclusion

In this paper, we presented the results of an experiment exploring the possibility to semi-automatically support the manual and expert work of correcting an error, be it by a teacher correcting a learner essay or by a linguist annotating errors in a learner corpus. We explored a two-steps ensemble approach that relies on several correction guessers to suggest candidate corrections for erroneous spans of text and uses a random forest classifier to select the valid ones.

Even though our current implementation would meet its original objective of supporting its targeted users only in a limited fashion, the preliminary results allow us to conclude that our approach has a fairly viable and encouraging potential.

We concluded the paper by discussing paths and challenges worth exploring or addressing.

References

Andrea Abel, Aivars Glaznieks, Lionel Nicolas, and Egon Stemle. 2016. An extended version of the koko german l1 learner corpus. In *Proceedings of Third Italian Conference on Computational Linguistics (CLiC-it 2016) & Fifth Evaluation Campaign of Natural Language Processing and Speech Tools for Italian. Final Workshop (EVALITA 2016)*.

Samuel Bell, Helen Yannakoudakis, and Marek Rei. 2019. Context is key: Grammatical error detection with contextual word representations. *arXiv preprint arXiv:1906.06593*.

Mengyun Chen, Tao Ge, Xingxing Zhang, Furu Wei, and Ming Zhou. 2020. Improving the efficiency of grammatical error correction with erroneous span detection and correction. *arXiv preprint arXiv:2010.03260*.

Yo Joong Choe, Jiyeon Ham, Kyubyong Park, and Yeoil Yoon. 2019. A neural grammatical error correction system built on better pre-training and sequential transfer learning. *arXiv preprint arXiv:1907.01256*.

Shamil Chollampatt and Hwee Tou Ng. 2017. Connecting the dots: Towards human-level grammatical error correction. In *Proceedings of the 12th Workshop on Innovative Use of NLP for Building Educational Applications*, pages 327–333, Copenhagen, Denmark. Association for Computational Linguistics.

Raj Dabre, Kehai Chen, Benjamin Marie, Rui Wang, Atsushi Fujita, Masao Utiyama, and Eiichiro Sumita. 2019. Nict's supervised neural machine translation systems for the wmt19 news translation task. In *Proceedings of the Fourth Conference on Machine Translation (Volume 2: Shared Task Papers, Day 1)*, pages 168–174.

Eibe Frank, Mark A. Hall, and Ian H. Witten. 2016. *The WEKA Workbench. Online Appendix for "Data Mining: Practical Machine Learning Tools and Techniques", Fourth Edition*. Morgan Kaufmann.

Ekaterina Garmash and Christof Monz. 2016. Ensemble learning for multi-source neural machine translation. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 1409–1418, Osaka, Japan. The COLING 2016 Organizing Committee.

Roman Grundkiewicz, Marcin Junczys-Dowmunt, and Kenneth Heafield. 2019. Neural grammatical error correction systems with unsupervised pre-training on synthetic data. In *Proceedings of the Fourteenth Workshop on Innovative Use of NLP for Building Educational Applications*, pages 252–263.

Tin Kam Ho. 1995. Random decision forests. In *Proceedings of 3rd international conference on document analysis and recognition*, volume 1, pages 278–282. IEEE.

Scott Jarvis and Aneta Pavlenko. 2008. *Crosslinguistic Influence in Language and Cognition*. Routledge.

Jianshu Ji, Qinlong Wang, Kristina Toutanova, Yongen Gong, Steven Truong, and Jianfeng Gao. 2017. A nested attention neural hybrid model for grammatical error correction. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 753–762, Vancouver, Canada. Association for Computational Linguistics.

Marcin Junczys-Dowmunt and Roman Grundkiewicz. 2016. Phrase-based machine translation is state-of-the-art for automatic grammatical error correction. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1546–1556, Austin, Texas. Association for Computational Linguistics.

Satoru Katsumata and Mamoru Komachi. 2020. Stronger baselines for grammatical error correction using pretrained encoder-decoder model. *arXiv preprint arXiv:2005.11849*.

Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondřej Bojar, Alexandra Constantin, and Evan Herbst. 2007. Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics Companion Volume Proceedings of the Demo and Poster Sessions*, pages 177–180, Prague, Czech Republic. Association for Computational Linguistics.

Preslav Nakov and Jörg Tiedemann. 2012. Combining word-level and character-level models for machine translation between closely-related languages. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 301–305, Jeju Island, Korea. Association for Computational Linguistics.

Jakub Náplava and Milan Straka. 2019. Grammatical error correction in low-resource scenarios. *arXiv preprint arXiv:1910.00353*.

Courtney Napoles, Keisuke Sakaguchi, and Joel Tetreault. 2017. JFLEG: A fluency corpus and benchmark for grammatical error correction. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 229–234, Valencia, Spain. Association for Computational Linguistics.

Nadezda Okinina and Lionel Nicolas. 2018. Towards smt-assisted error annotation of learner corpora. In *Fifth Italian Conference on Computational Linguistics*.

Kostiantyn Omelianchuk, Vitaliy Atrasevych, Artem Chernodub, and Oleksandr Skurzhanskyi. 2020.

Gector–grammatical error correction: Tag, not rewrite. *arXiv preprint arXiv:2005.12592*.

David Opitz and Richard Maclin. 1999. Popular ensemble methods: An empirical study. *J. Artif. Int. Res.*, 11(1):169–198.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Keisuke Sakaguchi, Kevin Duh, Matt Post, and Benjamin Van Durme. 2017. Robsut wrod reocginiton via semi-character recurrent neural network. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, AAAI'17, page 3281–3287. AAAI Press.

Allen Schmaltz, Yoon Kim, Alexander Rush, and Stuart Shieber. 2017. Adapting sequence models for sentence correction. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2807–2813, Copenhagen, Denmark. Association for Computational Linguistics.

Zheng Yuan and Ted Briscoe. 2016. Grammatical error correction using neural machine translation. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 380–386, San Diego, California. Association for Computational Linguistics.