Go Climb a Dependency Tree and Correct the Grammatical Errors

Longkai Zhang Houfeng Wang
Key Laboratory of Computational Linguistics (Peking University)
Ministry of Education, China
zhlongk@qq.com, wanghf@pku.edu.cn

Abstract

State-of-art systems for grammar error correction often correct errors based on word sequences or phrases. In this paper, we describe a grammar error correction system which corrects grammatical errors at tree level directly. We cluster all error into two groups and divide our system into two modules correspondingly: the general module and the special module. In the general module, we propose a TreeNode Language Model to correct errors related to verbs and nouns. The TreeNode Language Model is easy to train and the decoding is efficient. In the special module, two extra classification models are trained to correct errors related to determiners and prepositions. Experiments show that our system outperforms the state-of-art systems and improves the F_1 score.

1 Introduction

The task of grammar error correction is difficult yet important. An automatic grammar error correction system can help second language (L2) learners improve the quality of their writing. In recent years, there are various competitions devoted to grammar error correction, such as the HOO-2011(Dale and Kilgarriff, 2011), HOO-2012(Dale et al., 2012) and the CoNLL-2013 shared task (Ng et al., 2013). There has been a lot of work addressing errors made by L2 learners. A significant proportion of the systems for grammar error correction train individual statistical models to correct each special kind of error word by word and ignore error interactions. These methods assume no interactions between different kinds of grammatical errors. In real problem settings errors are correlated, which makes grammar error correction much more difficult.

Recent research begins to focus on the error interaction problem. For example, Wu and Ng (2013) decodes a global optimized result based on the individual correction confidence of each kind of errors. The individual correction confidence is still based on the noisy context. Rozovskaya and Roth (2013) uses a joint modeling approach, which considers corrections in phrase structures instead of words. For dependencies that are not covered by the joint learning model, Rozovskaya and Roth (2013) uses the results of Illinois system in the joint inference. These results are still at word level and are based on the noisy context. These systems can consider error interactions, however, the systems are complex and inefficient. In both Wu and Ng (2013) and Rozovskaya and Roth (2013), Integer Linear Programming (ILP) is used for decoding a global optimized result. In the worst case, the time complexity of ILP can be exponent.

In contrast, we think a better grammar error correction system should correct grammatical errors at sentence level directly and efficiently. The system should correct as many kinds of errors as possible in a generalized framework, while allowing special models for some kinds of errors that we need to take special care. We cluster all error into two groups and correspondingly divide our system into two modules: the general module and the special module. In the general module, our system views each parsed sentence as a dependency tree. The system generates correction candidates for each node on the dependency tree. The correction can be made on the dependency tree globally. In this module, nearly all replacement errors related to verb form, noun form and subject-verb agreement errors can be considered. In the special module, two extra classification models are used to correct the determiner errors and preposition errors . The classifiers are also trained at tree node level. We take special care of these two kinds

Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), pages 266–277,
October 25-29, 2014, Doha, Qatar. ©2014 Association for Computational Linguistics

of errors because these errors not only include replacement errors, but also include insertion and deletion errors. A classification model is more suitable for handling insertion and deletion errors. Besides, they are the most common errors made by English as a Second Language (ESL) learners and are much easier to be incorporated into a classification framework.

We propose a TreeNode Language Model (TNLM) to efficiently measure the correctness of selecting a correction candidate of a node in the general module. Similar to the existing statistical language models which assign a probability to a linear chain of words, our TNLM assigns correctness scores directly on each node on the dependency tree. We select candidates for each node to maximize the global correctness score and use these candidates to form the corrected sentence. The global optimized inference can be tackled efficiently using dynamic programming. Because the decoding is based on the whole sentence, error interactions can be considered. Our TNLM only needs to use context words related to each node on the dependency tree. Training a TreeNode language model costs no more than training ordinary language models on the same corpus. Experiments show that our system can outperform the state-of-art systems.

The paper is structured as follows. Section 1 gives the introduction. In section 2 we describe the task and give an overview of the system. In section 3 we describe the general module and in section 4 we describe the special module. Experiments are described in section 5. In section 6 related works are introduced, and the paper is concluded in the last section.

2 Task and System Overview

2.1 Task Description

The task of grammar error correction aims to correct grammatical errors in sentences. There are various competitions devoted to the grammar error correction task for L2 learners. The CoNLL-2013 shared task is one of the most famous, which focuses on correcting five types of errors that are commonly made by non-native speakers of English, including determiner, preposition, noun number, subject-verb agreement and verb form errors. The training data released by the task organizers come from the NUCLE corpus(Dahlmeier et al., 2013). This corpus contains essays written by ESL learners, which are then corrected by English teachers. The test data are 50 student essays. Details of the corpus are described in Ng et al. (2013).

2.2 System Architecture

In our system, lists of correction candidates are first generated for each word. We generate candidates for nouns based on their plurality. We generate candidates for verbs based on their tenses. Then we select the correction candidates that maximize the overall correctness score. An example process of correcting figure 1(a) is shown in table 1.

Correcting grammatical errors using local statistical models on word sequence is insufficient. The local models can only consider the contexts in a fixed window. In the example of figure 1(a), the context of the verb "is" is "that boy is on the", which sounds reasonable at first glance but is incorrect when considering the whole sentence. The limitation of local classifiers is that long distance syntax information cannot be incorporated within the local context. In order to effectively use the syntax information to get a more accurate correcting result, we think a better way is to tackle the problem directly at tree level to view the sentence as a whole. From figure 1(a) we can see that the node "is" has two children on the dependency tree: "books" and "on". When we consider the node "is", its context is "books is on", which sounds incorrect. Therefore, we can make better corrections using such context information on nodes.

Therefore, our system corrects grammatical errors on dependency trees directly. Because the correlated of words are more linked on trees than in a word sequence, the errors are more easier to be corrected on the trees and the agreement of different error types is guaranteed by the edges. We follow the strategy of treating different kinds of errors differently, which is used by lots of grammar error correction systems. We cluster the five types of errors considered in CoNLL-2013 into two groups and divide our system into two modules correspondingly.

- **The general module**, which is responsible for the verb form errors, noun number errors and subject-verb agreement errors. These errors are all replacement errors, which can be corrected by replacing the wrongly used word with a reasonable candidate word.

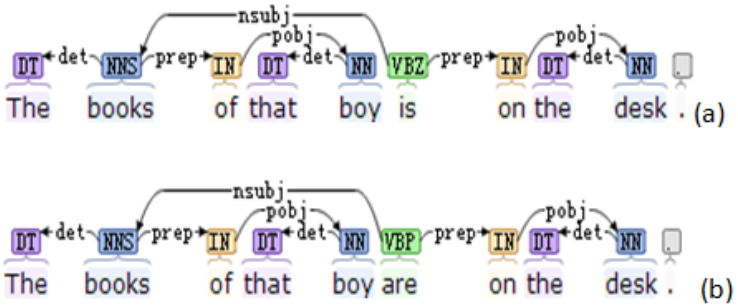

Figure 1: Dependency parsing results of (a) the original sentence "The books of that boy is on the desk ." (b) the corrected sentence.

Position	Original	Correction Candidates	Corrected
1	The	The	The
2	books	books, book	books
3	of	of	of
4	that	that	that
5	boy	boy, boys	boy
6	is	is,are,am,was,were,be,being,been	are
7	on	on	on
8	the	the	the
9	desk	desk, desks	desk
10	.	.	.

Table 1: An example of the "correction candidate generation and candidate selection" framework.

- **The special module**, where two classification models are used to correct the determiner errors and preposition errors at tree level. We take special care of these two kinds of errors because these errors include both replacement errors and insertion/deletion errors. Besides, they are the most common errors made by ESL learners and is much easier to be incorporated into a classification framework.

We should make it clear that we are not the first to use tree level correction models on ungrammatical sentences. Yoshimoto et al. (2013) uses a Treelet Language model (Pauls and Klein, 2012) to correct agreement errors. However, the performance of Treelet language model is not that good compared with the top-ranked system in CoNLL-2013. The reason is that the production rules in the Treelet language model are based on complex contexts, which will exacerbate the data sparseness problem. The "context" in Treelet language model also include words ahead of treelets, which are sometimes unrelated to the current node. In contrast, our TreeNode Language model only needs to consider useful context words related to each node on the dependency tree. To train a TreeNode language model costs no more than training ordinary language models on the same corpus.

2.3 Data Preparation

Our system corrects grammatical errors on dependency trees directly, therefore the sentences in training and testing data should have been parsed before being corrected. In our system, we use the Stanford parser[1] to parse the New York Times source of the Gigaword corpus[2], and use the parsed sentences as our training data. We use the original training data provided by CoNLL-2013 as the develop set to tune all parameters.

Some sentences in the news texts use a different writing style against the sentences written by ESL learners. For example, sentences written by ESL learners seldom include dialogues between people, while very often news texts include paragraphs such as "'I am frightened!' cried Tom". We use heuristic rules to eliminate the sentences in the Gigaword corpus that are less likely to appear in the ESL writing. The heuristic rules include delet-

[1]http://nlp.stanford.edu/software/lex-parser.shtml
[2]https://catalog.ldc.upenn.edu/LDC2003T05

ing sentences that are too short or too long[3], deleting sentences that contains certain punctuations such as quotation marks, or deleting sentences that are not ended with a period.

In total we select and parse 5 million sentences of the New York Times source of English newswire in the Gigaword corpus. We build the system and experiment based on these sentences.

3 The General Module

3.1 Overview

The general module aims to correct verb form errors, noun number errors and subject-verb agreement errors. Other replacement errors such as spelling errors can also be incorporated into the general module. Here we focus on verb form errors, noun number errors and subject-verb agreement errors only. Our general module views each sentence as a dependency tree. All words in the sentence form the nodes of the tree. Nodes are linked through directed edges, annotated with the dependency relations.

Before correcting the grammatical errors, the general module should generate correction candidates for each node first. For each node we use the word itself as its first candidate. Because the general module considers errors related to verbs and nouns, we generate extra correction candidates only for verbs and nouns. For verbs we use all its verb forms as its extra candidates. For example when considering the word "speaks", we use itself and {speak, spoke, spoken, speaking} as its correction candidates. For nouns we use its singular form and plural form as its extra correction candidates. For example when considering the word "dog", we use itself and "dogs" as its correction candidate. If the system selects the original word as the final correction, the sentence remains unchanged. But for convenience we still call the newly generated sentence "the corrected sentence".

In a dependency tree, the whole sentence s can be formulized as a list of production rules $r_1, ..., r_L$ of the form: $[r = head \rightarrow modifier_1, modifier_2 ...]$. An example of all production rules of figure 1(a) is shown in table 2. Because the production rules are made up of words, selecting a different correction candidate for only one node will result in a list of different production rules. For example, figure 1(b) selects the correction candidate "is" to replace the original "are". Therefore the production rules of figure 1(b) include [are → books, on], instead of [is → books, on] in figure 1(a).

books → The, of
of → boy
boy → that
is → books, on
on → desk
desk → the

Table 2: All the production rules in the example of figure 1(a)

The overall correctness score of s, which is $score(s)$, can be further decomposed into $\prod_{i=0}^{L} score(r_i)$. A reasonable score function should score the correct candidate higher than the incorrect one. Consider the node "is" in Figure 1(a), the production rule with head "is" is $[is \rightarrow books, on]$. Because the correction of "is" is "are", a reasonable scorer should have $score([is \rightarrow books, on]) < score([are \rightarrow books, on])$.

Given the formulation of sentence $s = [r_1, ..., r_L]$ and the candidates for each node, we are faced with two problems:

1. **Score Function**. Given a fixed selection of candidate for each node, how to compute the overall score of the dependency tree, i.e., $score(s)$. Because $score(s)$ is decomposed into $\prod_{i=0}^{L} score(r_i)$, the problem becomes finding a $score$ function to measure the correctness of each r given a fixed selection of candidates.

2. **Decoding**. Given each node a list of correction candidates and a reasonable score function $score(r)$ for the production rules, how to find the selection of candidates that maximize the overall score of the dependency tree.

For the first problem, we propose a TreeNode Language Model as the correctness measure of a fixed candidate selection. For the decoding problem, we use a dynamic programming method to efficiently find the correction candidates that maximize the overall score. We will describe the details in the following sections.

One concern is whether the automatically parsed trees are reliable for grammar error correction. We define "reliable" as follows. If we

[3] In our experiment, no less than 5 words and no more than 30 words.

change some words in original sentence into their reasonable correction candidates (e.g. change "is" to "are") but the structure of the dependency tree does not change (except the replaced word and its corresponding POS tag, which are definitely changed), then we say the dependency tree is reliable for this sentence. To verify this we randomly selected 1000 sentences parsed by the Stanford Parser. We randomly select the verbs and nouns and replace them with a wrong form. We parsed the modified sentences again and asked 2 annotators to examine whether the dependency trees are reliable for grammar error correction. We find that 99% of the dependency trees are reliable. Therefore we can see that the dependency tree can be used as the structure for grammar error correction directly.

3.2 TreeNode Language Model

In our system we use the score of TreeNode Language Model (TNLM) as the scoring function. Consider a node n on a dependency tree and assume n has K modifiers $C_1, ..., C_K$ as its child nodes. We define $Seq(n) = [C_1, ..., n, ..., C_K]$ as an ordered sub-sequence of nodes that includes the node n itself and all its child nodes. The order of the sub-sequence in $Seq(n)$ is sorted based on their position in the sentence. In this formulation, we can score the correctness of a production rule r by scoring the correctness of $Seq(n)$. Because $Seq(n)$ is a word sequence, we can use a language model to measure its correctness. The sub-sequences are not identical to the original text. Therefore instead of using ordinary language models, we should train special language models using the sub-sequences to measure the correctness of a production rule.

Take the sentence in figure 2 as an example. When considering the node "is" in the word sequence, it is likely to be corrected into "are" because it appear directly after the plural noun "parents". However, by the definition above, the subsequence corresponding to the node "damaged" is "car is damaged by ". In such context, the word "is" is less likely to be changed to "are". From the example we can see that the sub-sequence is suitable to be used to measure the correctness of a production rule. From this example we can also find that the sub-sequences are different with ordinary sentences, because ordinary sentences are less likely to end with "by".

Table 3 shows all the sub-sequences in the example of figure 2. If we collect all the subsequences in the corpus to form a new subsequence corpus, we can train a language model based on the new sub-sequence corpus. This is our TreeNode Language Model. One advantage of TLM is that once we have generated the subsequences, we can train the TLM in the same way as we train ordinary language models. Besides, the TLM is not limited to a fixed smoothing method. Any smoothing methods for ordinary language models are applicable for TLM.

Node	Sub-sentence
The	The
car	The car of
of	of parents
my	my
parents	my parents
is	is
damaged	car is damaged by
by	by storm
the	the
storm	the storm

Table 3: All the sub-sentences in the example of figure 2

In our system we train the TLM using the same way as training tri-gram language model. For a sub-sequence $S = w_0...w_L$, we calculate $P(S) = \prod_{i=0}^{L} P(w_i|w_{i-1}w_{i-2})$. The smoothing method we use is interpolation, which assumes the final $P'(w_i|w_{i-1}w_{i-2})$ of the tri-gram language model follows the following decomposition:

$$\begin{aligned} P'(w_i|w_{i-1}w_{i-2}) =& \lambda_1 P(w_i|w_{i-1}w_{i-2}) \\ &+ \lambda_2 P(w_i|w_{i-1}) \\ &+ \lambda_3 P(w_i) \end{aligned} \quad (1)$$

where λ_1, λ_2 and λ_3 are parameters sum to 1. The parameters λ_1, λ_2 and λ_3 are estimated through EM algorithm(Baum et al., 1970; Dempster et al., 1977; Jelinek, 1980).

3.3 Decoding

The decoding problem is to select one correction candidate for each node that maximizes the overall score of the corrected sentence. When the sentence is long and contains many verbs and nouns, enumerating all possible candidate selections is time-consuming. We use a bottom-up dynamic

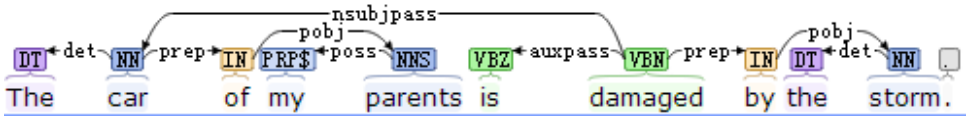

Figure 2: A illustrative sentence for TreeNode Language Model.

programming approach to find the maximized corrections within polynomial time complexity.

For a node n with L correction candidates $n_1, ...n_L$ and K child nodes $C_1, ..., C_K$, we define $n.scores[i]$ as the maximum score if we choose the ith candidate n_i for n. Because we decode from leaves to the root, $C_1.scores, ..., C_K.scores$ have already been calculated before we calculate $n.scores$.

We assume the sub-sequence $Seq(n_i) = [C_1, ..., C_M, n_i, C_{M+1}, ..., C_K]$ without loss of generality, where $C_1, .., C_M$ are the nodes before n_i and $C_{M+1}, ..., C_K$ are the nodes after n_i.

We define $c_{i,j}$ as the jth correction candidate of child node C_i. Given a selection of candidates for each child node $seq = [c_{1,j_1}, ..., c_{M,j_M}, n_i, c_{M+1,j_{M+1}}, ..., c_{K,j_K}]$, we can calculate $score(seq)$ as:

$$score(seq) = TNLM(seq) \prod_{i=1}^{K} C_i.scores[j_i] \quad (2)$$

where $TNLM(seq)$ is the TreeNode Language Model score of seq. Then, $n.scores[i]$ is calculated as:

$$n.scores[i] = \max_{\forall seq} score(seq) \quad (3)$$

Because seq is a word sequence, the maximization can be efficiently calculated using Viterbi algorithm (Forney Jr, 1973). To be specific, the Viterbi algorithm uses the transition scores and emission scores as its input. The transition scores in our model are the tri-gram probabilities from our tri-gram TNLM. The emission scores in our model are the candidate scores of each child: $C_1.scores, ..., C_K.scores$, which have already been calculated.

After the bottom-up calculation, we only need to look into the "ROOT" node to find the maximum score of the whole tree. Similar to the Viterbi algorithm, back pointers should be kept to find which candidate is selected for the final corrected sentence. Detailed decoding algorithm is shown in table 4.

```
Function decode(Node n)
  if n is leaf
    set n.scores uniformly
    return
  for each child c of n
    decode(c)
  calculating n.scores using Viterbi
End Function
BEGIN
  decode(ROOT)
  find the maximum score for the tree and backtrack all candidates
END
```

Table 4: The Decoding algorithm

In the real world implementations, we add a controlling parameter for the confidence of the correctness of the inputs. We multiply λ on $P(w_0|w_{-2}w_{-1})$ of the tri-gram TNLM if the correcting candidate w_0 is the same word in the original input. λ is larger than 1 to "emphasis" the confidence of the original word because the most of the words in the inputs are correct. The value of λ can be set using the development data.

3.4 The Special Module

The special module is designed for determiner errors and preposition errors. We take special care of these two kinds of errors because these errors include insertion and deletion errors, which cannot be corrected in the general module. Because there is a fixed number of prepositions and determiners, these two kinds of errors are much easier to be incorporated into a classification framework. Besides, they are the most common errors made by ESL learners and there are lots of previous works that leave valuable guidance for us to follow.

Similar to many previous state-of-art systems, we treat the correction of determiner errors and preposition errors as a classification problem. Although some previous works (e.g. Rozovskaya et al. (2013)) use NPs and the head of NPs as

features, they are basically local classifiers making predictions on word sequences. Difference to the local classifier approaches, we make predictions on the nodes of the dependency tree directly. In our system we correct determiner errors and preposition errors separately.

For the determiner errors, we consider the insertion, deletion and replacement of articles (i.e. 'a', 'an' and 'the'). Because the articles are used to modify nouns in the dependency trees, we can classify based on noun nodes. We give each noun node (node whose POS tag is noun) a label to indicate which article it should take. We use left position (LP) and right position (RP) to specify the position of the article. The article therefore locates between LP and RP. If a noun node already has an article as its modifier, then LP will be the position directly ahead of the article. In this case, $RP = LP + 2$. If an insertion is needed, the RP is the position of the first child node of the noun node. In this case $LP = RP - 1$. With this notation, detailed feature templates we use to correct determiner errors are listed in table 5. In our model we use 3 labels: 'a', 'the' and '$\emptyset$'. We use 'a', 'the' to represent a noun node should be modified with 'a' or ''the' correspondingly. We use ''$\emptyset$' to indicate that no article is needed for the noun node. We use rule-based method to distinguish between "a" and "an" as a post-process.

For the preposition errors, we only consider deletion and replacement of an existing preposition. The classification framework is similar to determiner errors. We consider classification on preposition nodes (nodes whose POS tag is preposition). We use prepositions as labels to indicate which preposition should be used. and use "$\emptyset$" to denote that the preposition should be deleted. We use the same definition of LP and RP as the correction of determiner errors. Detailed feature templates we use to correct preposition errors are listed in table 6. Similar to the previous work(Xing et al., 2013), we find that adding more prepositions will not improve the performance in our experiments. Thus we only consider a fixed set of prepositions: {in, for, to, of, on}.

Previous works such as Rozovskaya et al. (2013) show that Naive Bayes model and averaged perceptron model show better results than other classification models. These classifiers can give a reasonably good performance when there are limited amount of training data. In our system, we use large amount of automatically generated training data based on the parsed Gigaword corpus instead of the limited training data provided by CoNLL-2013.

Take generating training data for determiner errors as an example. We generate training data based on the parsed Gigaword corpus C described in section 2. Each sentence S in C is a dependency tree T. We use each noun node N on T as one training instance. If N is modified by "the", its label will be "THE". If N is modified by "a" or "an", its label will be "A". Otherwise its label will be "NULL". Then we just omit the determiner modifier and generate features based on table 5. Generating training data for preposition errors is the same, except we use preposition nodes instead of noun nodes.

By generating training instances in this way, we can get large amount of training data. Therefore we think it is a good time to try different classification models with enough training data. We experiment on Naive Bayes, Averaged Perceptron, SVM and Maximum Entropy models (ME) in a 5-fold cross validation on the training data. We find ME achieves the highest accuracy. Therefore we use ME as the classification model in our system.

4 Experiment

4.1 Experiment Settings

In the experiments, we use our parsed Gigaword corpus as the training data, use the training data provided by CoNLL-2013 as the develop data, and use the test data of CoNLL-2013 as test data directly. In the general module, the training data is used for the training of TreeNode Language Model. In the special module, the training data is used for training individual classification models.

We use the M2 scorer (Dahlmeier and Ng, 2012b) provided by the organizer of CoNLL-2013 for the evaluation of our system. The M2 scorer is widely used as a standard scorer in previous systems. Because we make comparison with the state-of-art systems on the CoNLL-2013 corpus, we use the same evaluation metric F_1 score of M2 scorer as the evaluation metric.

In reality, some sentences may have more than one kind of possible correction. As the example in "The books of that boy is on the desk.", the corresponding correction can be either "The books of that boy are on the desk." or "The book of that boy is on the desk.". The gold test data can only con-

Word Features	w_{LP} , w_{LP-1}, w_{LP-2}, w_{RP}, w_{RP+1}, w_{RP+2}, $w_{LP-2}w_{LP-1}$, $w_{LP-1}w_{LP}$, $w_{LP}w_{RP}$, $w_{RP}w_{RP+1}$, $w_{RP+1}w_{RP+2}$, $w_{LP-2}w_{LP-1}w_{LP}$, $w_{LP-1}w_{LP}w_{RP}$, $w_{LP}w_{RP}w_{RP+1}$, $w_{RP}w_{RP+1}w_{RP+2}$
Noun Node Features	NN, $w_{LP}NN$, $w_{LP-1}w_{LP}NN$, $w_{LP-2}w_{LP-1}w_{LP}NN$
Father/Child Node Features	Fa, $w_{RP}Fa$, $w_{RP}w_{RP+1}Fa$, $w_{RP}w_{RP+1}w_{RP+2}Fa$, $Fa\&Ch$

Table 5: Feature templates for the determiner errors. w_i is the word at the ith position. NN is the current noun node. Fa is the father node of the current preposition node. Ch is a child node of the current preposition node.

Word Features	w_{LP} , w_{LP-1}, w_{LP-2}, w_{RP}, w_{RP+1}, w_{RP+2}, $w_{LP-2}w_{LP-1}$, $w_{LP-1}w_{LP}$, $w_{LP}w_{RP}$, $w_{RP}w_{RP+1}$, $w_{RP+1}w_{RP+2}$, $w_{LP-2}w_{LP-1}w_{LP}$, $w_{LP-1}w_{LP}w_{RP}$, $w_{LP}w_{RP}w_{RP+1}$, $w_{RP}w_{RP+1}w_{RP+2}$
Father/Child Node Features	Fa, $w_{RP}Fa$, $w_{RP}w_{RP+1}Fa$, $w_{RP}w_{RP+1}w_{RP+2}Fa$, $Fa\&Ch$

Table 6: Feature templates for preposition errors. w_i is the word at the ith position. Fa is the father node of the current preposition node. Ch is a child node of the current preposition node.

sider a small portion of possible answers. To relieve this, the CoNLL-2013 shared task allows all participating teams to provide alterative answers if they believe their system outputs are also correct. These alterative answers form the "Revised Data" in the shared task, which indeed help evaluate the outputs of the participating systems. However, the revised data only include alterative corrections from the participating teams. Therefore the evaluation is not that fair for future systems. In our experiment we only use the original test data as the evaluation dataset.

4.2 Experiment Results

We first show the performance of each stage of our system. In our system, the general module and the special module correct grammar errors consequently. Therefore in table 7 we show the performance when each component is added to the system.

Method	P	R	F1 score
TNLM	33.96%	17.71%	23.28%
+Det	32.83%	38.28%	35.35%
+Prep	32.64%	39.20%	35.62%

Table 7: Results of each stage in our system. TNLM is the general module. "+Det" is the system containing the general module and determiner part of special module."+Prep" is the final system

We evaluate the effect of using TreeNode language model for the general module. We compare the TNLM with ordinary tri-gram language model. We use the same amount of training data and the same smoothing strategy (i.e. interpolation) for both of them. Table 8 shows the comparison. The TNLM can improve the F_1 by $+2.1\%$.

Method	P	R	F1 score
Ordinary LM	29.27%	16.68%	21.27%
Our TNLM	**33.96%**	**17.71%**	**23.28%**

Table 8: Comparison for the general module between TNLM and ordinary tri-gram language model on the test data.

Based on the result of the general module using TNLM, we compare our tree level special module against the local classification approach. The special module of our system makes predictions on the dependency tree directly, while local classification approaches make predictions on linear chain of words and decide the article of a noun Phrase or the preposition of a preposition phrase. We use the same word level features for the two approaches except for the local classifiers we do not add tree level features. Table 9 shows the comparison.

When using the parsed Gigaword texts as training data, the quality of the sentences we select will influence the result. For comparison, we randomly select the same amount of sentences from the same source of Gigaword and parse them as

Method	P	R	F1 score
Local Classifier	26.38%	39.14%	31.51%
Our Tree-based	**32.64%**	**39.20%**	**35.62%**

Table 9: Comparison for the special module on the test data. The input of the special module is the sentences corrected by the TNLM in the general module.

a alterative training set. Table 10 shows the comparison between random chosen training data and the selected training data of our system. We can see that the data selection (cleaning) procedure is important for the improvement of system $F1$.

Method	P	R	F1 score
Random	31.89%	35.85%	33.75%
Selected	**32.64%**	**39.20%**	**35.62%**

Table 10: Comparison of training using random chosen sentences and selected sentences.

Method	F1 score
Rozovskaya et al. (2013)	31.20%
Kao et al. (2013)	25.01%
Yoshimoto et al. (2013)	22.17%
Rozovskaya and Roth (2013)	35.20%
Our method	**35.62**%

Table 11: Comparison of $F1$ of different systems on the test data .

4.3 Comparison With Other Systems

We also compare our system with the state-of-art systems. The first two are the top-2 systems at CoNLL-2013 shared task : Rozovskaya et al. (2013) and Kao et al. (2013). The third one is the Treelet Language Model in Yoshimoto et al. (2013). The fourth one is Rozovskaya and Roth (2013), which until now shows the best performance. The comparison on the test data is shown in table 11.

In CoNLL-2013 only 5 kinds of errors are considered. Our system can be slightly modified to handle the case where other errors such as spelling errors should be considered. In that case, we can modify the candidate generation of the general module. We only need to let the generate correction candidates be any possible words that are similar to the original word, and run the same decoding algorithm to get the corrected sentence. As a comparison, the ILP systems should add extra scoring system to score extra kind of errors.

5 Related Works

Early grammatical error correction systems use the knowledge engineering approach (Murata and Nagao, 1994; Bond et al., 1996; Bond and Ikehara, 1996; Heine, 1998). However, manually designed rules usually have exceptions. Therefore, the machine learning approach has become the dominant approach recently. Previous machine learning approaches typically formulates the task as a classification problem. Of all the errors, determiner and preposition errors are the two main research topics (Knight and Chander, 1994; AEHAN et al., 2006; Tetreault and Chodorow, 2008; Dahlmeier and Ng, 2011). Features used in the classification models include the context words, POS tags, language model scores (Gamon, 2010), and tree level features (Tetreault et al., 2010). Models used include maximum entropy (AEHAN et al., 2006; Tetreault and Chodorow, 2008), averaged perceptron, Naive Bayes (Rozovskaya and Roth, 2011), etc. Other errors such as verb form and noun number errors also attract some attention recently (Liu et al., 2010; Tajiri et al., 2012).

Recent research efforts have started to deal with correcting different errors jointly (Gamon, 2011; Park and Levy, 2011; Dahlmeier and Ng, 2012a; Wu and Ng, 2013; Rozovskaya and Roth, 2013). Gamon (2011) uses a high-order sequential labeling model to detect various errors. Park and Levy (2011) models grammatical error correction using a noisy channel model. Dahlmeier and Ng (2012a) uses a beam search decoder, which iteratively corrects to produce the best corrected output. Wu and Ng (2013) and Rozovskaya and Roth (2013) use ILP to decode a global optimized result. The joint learning and joint inference are still at word/phrase level and are based on the noisy context. In the worst case, the time complexity of ILP can reach exponent. In contrast, our system corrects grammar errors at tree level directly, and the decoding is finished with polynomial time complexity.

6 Conclusion and Future work

In this paper we describe our grammar error correction system which corrects errors at tree level directly. We propose a TreeNode Language Model and use it in the general module to correct errors related to verbs and nouns. The TNLM is easy to

train and the decoding of corrected sentence is efficient. In the special module, two extra classification models are trained to correct errors related to determiners and prepositions at tree level directly. Because our current method depends on an automatically parsed corpus, future work may include applying some additional filtering (e.g. Mejer and Crammer (2012)) of the extended training set according to some confidence measure of parsing accuracy.

Acknowledgments

This research was partly supported by National Natural Science Foundation of China (No.61370117,61333018), Major National Social Science Fund of China (No.12&ZD227) and National High Technology Research and Development Program of China (863 Program) (No. 2012AA011101). The contact author of this paper, according to the meaning given to this role by Key Laboratory of Computational Linguistics, Ministry of Education, School of Electronics Engineering and Computer Science, Peking University, is Houfeng Wang. We thank Longyue Wang and the reviewers for their comments and suggestions.

References

AEHAN, N., Chodorow, M., and LEACOCK, C. L. (2006). Detecting errors in english article usage by non-native speakers.

Baum, L. E., Petrie, T., Soules, G., and Weiss, N. (1970). A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains. *The annals of mathematical statistics*, pages 164–171.

Bond, F. and Ikehara, S. (1996). When and how to disambiguate?—countability in machine translation—. In *International Seminar on Multimodal Interactive Disambiguation: MIDDIM-96*, pages 29–40. Citeseer.

Bond, F., Ogura, K., and Kawaoka, T. (1996). Noun phrase reference in japanese-to-english machine translation. *arXiv preprint cmp-lg/9601008*.

Dahlmeier, D. and Ng, H. T. (2011). Grammatical error correction with alternating structure optimization. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 915–923. Association for Computational Linguistics.

Dahlmeier, D. and Ng, H. T. (2012a). A beam-search decoder for grammatical error correction. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 568–578. Association for Computational Linguistics.

Dahlmeier, D. and Ng, H. T. (2012b). 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. Association for Computational Linguistics.

Dahlmeier, D., Ng, H. T., and Wu, S. M. (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.

Dale, R., Anisimoff, I., and Narroway, G. (2012). Hoo 2012: A report on the preposition and determiner error correction shared task. In *Proceedings of the Seventh Workshop on Building Educational Applications Using NLP*, pages 54–62. Association for Computational Linguistics.

Dale, R. and Kilgarriff, A. (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.

Dempster, A. P., Laird, N. M., Rubin, D. B., et al. (1977). Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal statistical Society*, 39(1):1–38.

Forney Jr, G. D. (1973). The viterbi algorithm. *Proceedings of the IEEE*, 61(3):268–278.

Gamon, M. (2010). Using mostly native data to correct errors in learners' writing: a meta-classifier approach. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 163–171. Association for Computational Linguistics.

Gamon, M. (2011). High-order sequence modeling for language learner error detection. In *Proceedings of the 6th Workshop on Innovative Use of NLP for Building Educational Applications*, pages 180–189. Association for Computational Linguistics.

Heine, J. E. (1998). Definiteness predictions for japanese noun phrases. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics-Volume 1*, pages 519–525. Association for Computational Linguistics.

Jelinek, F. (1980). Interpolated estimation of markov source parameters from sparse data. *Pattern recognition in practice*.

Kao, T.-h., Chang, Y.-w., Chiu, H.-w., Yen, T.-H., Boisson, J., Wu, J.-c., and Chang, J. S. (2013). Conll-2013 shared task: Grammatical error correction nthu system description. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning: Shared Task*, pages 20–25, Sofia, Bulgaria. Association for Computational Linguistics.

Knight, K. and Chander, I. (1994). Automated postediting of documents. In *AAAI*, volume 94, pages 779–784.

Liu, X., Han, B., Li, K., Stiller, S. H., and Zhou, M. (2010). Srl-based verb selection for esl. In *Proceedings of the 2010 conference on empirical methods in natural language processing*, pages 1068–1076. Association for Computational Linguistics.

Mejer, A. and Crammer, K. (2012). Are you sure? confidence in prediction of dependency tree edges. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 573–576, Montréal, Canada. Association for Computational Linguistics.

Murata, M. and Nagao, M. (1994). Determination of referential property and number of nouns in japanese sentences for machine translation into english. *arXiv preprint cmp-lg/9405019*.

Ng, H. T., Wu, S. M., Wu, Y., Hadiwinoto, C., and Tetreault, J. (2013). The conll-2013 shared task on grammatical error correction. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning: Shared Task*, pages 1–12, Sofia, Bulgaria. Association for Computational Linguistics.

Park, Y. A. and Levy, R. (2011). Automated whole sentence grammar correction using a noisy channel model. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 934–944. Association for Computational Linguistics.

Pauls, A. and Klein, D. (2012). Large-scale syntactic language modeling with treelets. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1*, pages 959–968. Association for Computational Linguistics.

Rozovskaya, A., Chang, K.-W., Sammons, M., and Roth, D. (2013). The university of illinois system in the conll-2013 shared task. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning: Shared Task*, pages 13–19, Sofia, Bulgaria. Association for Computational Linguistics.

Rozovskaya, A. and Roth, D. (2011). Algorithm selection and model adaptation for esl correction tasks. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 924–933. Association for Computational Linguistics.

Rozovskaya, A. and Roth, D. (2013). Joint learning and inference for grammatical error correction. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 791–802, Seattle, Washington, USA. Association for Computational Linguistics.

Tajiri, T., Komachi, M., and Matsumoto, Y. (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.

Tetreault, J., Foster, J., and Chodorow, M. (2010). Using parse features for preposition selection and error detection. In *Proceedings of the acl 2010 conference short papers*, pages 353–358. Association for Computational Linguistics.

Tetreault, J. R. and Chodorow, M. (2008). The ups and downs of preposition error detec-

tion in esl writing. In *Proceedings of the 22nd International Conference on Computational Linguistics-Volume 1*, pages 865–872. Association for Computational Linguistics.

Wu, Y. and Ng, H. T. (2013). Grammatical error correction using integer linear programming. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1456–1465, Sofia, Bulgaria. Association for Computational Linguistics.

Xing, J., Wang, L., Wong, D. F., Chao, L. S., and Zeng, X. (2013). Um-checker: A hybrid system for english grammatical error correction. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning: Shared Task*, pages 34–42, Sofia, Bulgaria. Association for Computational Linguistics.

Yoshimoto, I., Kose, T., Mitsuzawa, K., Sakaguchi, K., Mizumoto, T., Hayashibe, Y., Komachi, M., and Matsumoto, Y. (2013). Naist at 2013 conll grammatical error correction shared task. *CoNLL-2013*, 26.