Automatic Article Restoration

John Lee
Spoken Language Systems
MIT Computer Science and Artificial Intelligence Laboratory
Cambridge, MA 02139, USA
jsylee@sls.csail.mit.edu

Abstract

One common mistake made by non-native speakers of English is to drop the articles *a*, *an*, or *the*. We apply the log-linear model to automatically restore missing articles based on features of the noun phrase. We first show that the model yields competitive results in article generation. Further, we describe methods to adjust the model with respect to the initial quality of the sentence. Our best results are 20.5% article error rate (insertions, deletions and substitutions) for sentences where 30% of the articles have been dropped, and 38.5% for those where 70% of the articles have been dropped.

1 Introduction

An English noun phrase (NP) may contain a determiner, such as *this*, *that*, *a*, *an* or *the*, which specifies the reference of its head. The two most common of these determiners, *a*/*an* and *the*, are also known as articles. Broadly speaking, *the* indicates that the head refers to someone or something that is uniquely defined; *a*/*an*, or the absence of any articles, indicates that it is a general concept.

Many languages do not have any articles. Native speakers of these languages often have difficulty choosing appropriate English articles, and tend to underuse them. Our general goal is to automatically correct the use of articles in English sentences written by non-native speakers. In this paper, we describe methods for a more specific task: restoring missing articles.

2 Related Work

The **article generation** task could be viewed as a classification problem, whose input is a set of features drawn from the context of an NP, and whose output is the most likely article for that NP. The context features are typically extracted from the syntactic parse tree of a sentence.

(Heine, 1998) takes a Japanese NP as input, and classifies it as either definite or indefinite. A hierarchy of rules, ordered by their priorities, are hand-crafted. These rules involve the presence or absence of honorifics, demonstratives, possessives, counting expressions, and a set of verbs and postpositions that provide strong hints. In the appointment scheduling domain, 79.5% of the NPs are classified with an accuracy of 98.9%. The rest are classified by searching for its referent in the discourse context.

(Knight and Chander, 1994) uses decision trees to pick either *a*/*an* or *the* for NPs extracted from the Wall Street Journal (WSJ). There are over 30000 features in the trees, including lexical features (e.g., the two words before and after the NP) and abstract features (e.g., the word after the head noun is a past tense verb). By classifying the more frequent head nouns with the trees, and guessing *the* for the rest, the overall accuracy is 78%.

(Minnen et al., 2000) applies a memory-based learning approach to choose between *a*/*an*, *the* and *null*. Their features are drawn from two sources: first, from the Penn Treebank, such as the NP head and its part-of-speech (POS) and functional tags, the category and functional tags of the constituent embedding the NP, and other determiners in the NP; and second, from a Japanese-to-English translation system, such as the countability preference and semantic class of the NP head. The best result is 83.6% accuracy.

3 Approach

The article generation task constitutes one component of the **article correction** task. The other component is a natural language parser that maps an input sentence to a parse tree, from which context features of NPs are extracted. In addition, the article correction task needs to address two issues:

- Ideally, the parse tree of an input sentence with inappropriate articles should be identical (except, of course, the leaves for the articles) to that of the

equivalent correct sentence. However, a natural language parser, trained on grammatical sentences, does not perform as well on sentences with inappropriate articles. It might not be able to identify all NPs accurately. We evaluate this problem in §4.4. Further, the context features of the NPs might be distorted. The performance of the article generator is likely to suffer. We measure this effect in §4.5.

- The input sentence may already contain some articles. If the sentence is of high 'quality', one should be conservative in making changes to its articles. We characterize 'quality' using a 3×3 confusion matrix. The articles on the rows are the correct ones; those on the columns are the ones actually used in the sentence. For example, if a sentence has the matrix

	a	*null*	*the*
a	0.1	0.9	0
null	0	1	0
the	0	0.6	0.4

  then the article *the* is correctly used in the sentence with a 40% chance, but is mistakenly dropped (i.e., substituted with *null*) with a 60% chance. If one could accurately estimate the underlying confusion matrix of a sentence, then one could judiciously use the existing articles as a factor when generating articles.

  For the **article restoration** task, we assume that articles may be dropped, but no unnecessary articles are inserted, and the articles *the* and *a* are not confused with each other. In other words, the four zero entries in the matrix above are fixed. We report experiments on article restoration in §4.6.

3.1 Features

Our context features are drawn from two sources: the output of Model 3 of Collins' statistical natural language parser (Collins, 1999), and WordNet Version 2.0. For each base NP in the parse tree, we extract 15 categories of syntactic and semantic features. As an example, the sentence *Pierre Vinken, 61 years old, will join the board as a nonexecutive director Nov. 29* is parsed as:

... the/DT board/NN)
(PP as/IN
(NPB a/DT nonexecutive/JJ director/NN))
(NPB Nov./NNP 29/CD ...

From this parse tree the following features are extracted for the base NP *a nonexecutive director*:

Article* The correct article, which may be *the*, *null*, or *a* (covering both *a* and *an*).

Article (*a*) The article in the original sentence.

Head (*director*) The root form of the head of the NP. A number is rewritten as <*number*>. The head is determined using the rules in (Collins, 1999), except for possessive NPs. The head of a possessive NP is *'s*, which is not indicative of its article preference. Instead, we use the second best candidate for NP head.

Number (*singular*) If the POS tag of the NP head is *NN* or *NNP*, the number of the head is *singular*; if the tag is *NNS* or *NNPS*, it is *plural*; for all other tags, it is *n/a*.

Head POS (*NN*) The POS tag of the NP head. Any information about the head's number is hidden; *NNS* is re-written as *NN*, and *NNPS* as *NNP*.

Parent (*PP*) The category of the parent node of the NP.

Non-article determiner (*null*) A determiner other than *a* or *the* in the NP.

Words before head (*nonexecutive*) Words inside the NP that precede the head, excluding determiners.

Words after head (*null*) Words inside the NP that follow the head, excluding determiners.

POS of words before head (*JJ*) The POS tags of words inside the NP that precede the head, excluding determiners.

POS of words after head (*null*) The POS tags of words inside the NP that follow the head, excluding determiners.

Words before NP (*board, as*) The two words preceding the base NP. This feature may be *null*.

Words after NP (*Nov, <number>*) The two words following the base NP. This feature may be *null*.

Hypernyms ({*entity*}, {*object, physical object*}, ..., {*head, chief, top dog*}, {*administrator, decision maker*}) Each synset in the hierarchy of hypernyms for the head in WordNet is considered a feature. We do not attempt any sense disambiguation, but always use the hypernyms for the first sense.

Referent (*no*) If the same NP head appears in one of the 5 previous sentences, then *yes*; otherwise, *no*.

3.2 Log-linear Model

We use the log-linear model (Ratnaparkhi, 1998), which has the maximum entropy property, to estimate the conditional probabilities of each value of the **Article*** feature, given any combination of features. This model is able

to incorporate all these features, despite their interdependence, in a straightforward manner. Furthermore, unlike in decision trees, there is no need to partition the training data, thereby alleviating the data sparseness problem.

In this model, the **Article*** feature is paired up with each of the other features to form contextual predicates (also called "features" in (Ratnaparkhi, 1998)). Thus, our example sentence has the following predicates:

(Article = a) & (Article = a)*
(Article = a) & (Head = director)*
(Article = a) & (Head POS = NN)*
...

4 Experiments

4.1 Training Sets for Article Restoration

We ran Ratnaparkhi's MXPOST part-of-speech tagger and Model 3 of Collins' parser on the text in sections 00 to 21 of the Penn Treebank-3. We then extracted all base NPs and their features from the parser's output.[1] There are about 260000 base NPs. The distribution of the articles in this set is roughly 70.5% *null*, 20% *the* and 9.5% *a*.

The articles in the original sentences were initially assigned to both the **Article*** and **Article** features. This would imply a very high quality for the input sentences, in the sense that their articles were extremely likely to be correct. As a result, the model would be overly conservative about inserting new articles. To simulate varying qualities of input sentences, we perturbed the **Article** feature with two different confusion matrices, resulting in the following training sets:

- TRAINDROP70: The **Article** feature is perturbed according to the confusion matrix

$$\begin{pmatrix} 0.3 & 0.7 & 0 \\ 0 & 1 & 0 \\ 0 & 0.7 & 0.3 \end{pmatrix}$$

  That is, 70% of the feature *(Article = the)*, and 70% of the feature *(Article = a)*, are replaced with the feature *(Article = null)*. The rest are unchanged.

  This set trains the model to aim to insert enough articles such that the initial number of articles in a sentence would constitute about 30% of the final number of articles.

- TRAINDROP30: The **Article** feature is perturbed according to the confusion matrix

$$\begin{pmatrix} 0.7 & 0.3 & 0 \\ 0 & 1 & 0 \\ 0 & 0.3 & 0.7 \end{pmatrix}$$

  That is, 30% of *(Article = the)* and 30% of *(Article = a)* are replaced with *(Article = null)*. Upon seeing a *null* in an input sentence, all else being equal, TRAINDROP30 should be less predisposed than TRAINDROP70 to change it to *the* or *a*. In other words, the weight of *(Article* = the) & (Article = null)* and *(Article* = a) & (Article = null)* should be heavier in TRAINDROP70 than TRAINDROP30.

Contextual predicates that were true in less than 5 base NPs in the training sets were deemed unreliable and rejected. The weight for each predicate was initialized to zero, and then trained by iterative scaling.

After training on TRAINDROP30 for 1500 rounds, the ten heaviest weights were:

(Article = the) & (Head = the)*[2]
(Article = a) & (Word before head = lot)*
(Article = the) & (Head = Netherlands)*
(Article = the) & (Head = Beebes)*
(Article = a) & (Word before head = million)*
(Article = a) & (Hypernym = {struggle, battle})*
(Article = the) & (Word before head = year-before)*[3]
(Article = a) & (Word before head = dozen)*
(Article = a) & (Word before head = restated)*
(Article = the) & (Head = wound)*

Notice that two features, **Head** and **Word before head**, dominated the top 10 weights.

4.2 Training Sets for Article Generation

We created three additional training sets which omit the **Article** feature. In other words, the articles in input sentences would be ignored. These sets were used in the article generation experiments.

- TRAINGEN$_{base}$: This set uses only four features, **Article***, **Head**, **Number** and **Head POS**.

[1] Since Collins' parser was trained on sections 02 to 21 of the same treebank, the accuracy of our context features is higher than what we would expect from other texts. Our motivation for using the text of the Penn Treebank is to facilitate comparison between our article generation results and those reported in (Knight and Chander, 1994) and (Minnen et al., 2000), both of which read context features directly from the Penn Treebank.

[2] The article *the* as head of an NP is due to incorrect parses. An example is the sentence *Mr. Nixon, the most prominent American to come to China since ....* The parse had an *S* parent dominating a base NP, which contained *the* alone, and an adjective phrase, which contained *most prominent American* and so forth.

[3] The word *year-before* is used as an adjective in the NP, such as *the year-before $33 million*.

Accuracy Rate	DROP0	DROP30	DROP70
TRAINGEN	87.7%	82.5%	76.4%
TRAINGEN_{Minnen}	82.4%	79.5%	75.8%
TRAINGEN_{base}	80.1%	78.6%	76.9%

Table 1: Accuracy rate in article generation

Article	*null* generated	*the* generated	*a* generated
null	9647	324	124
the	656	1898	228
a	167	249	878

Table 2: Contingency table for article generation using TRAINGEN on DROP0

- TRAINGEN_{Minnen}: This set uses the subset of our features that were also used in (Minnen et al., 2000). These include all the features in TRAINGEN_{base}, plus **Parent** and **Non-article determiner**.
- TRAINGEN: This set uses our full set of features.

4.3 Test Sets

We generated four test sets from the text in section 23 of the Penn Treebank-3 by dropping 70%, 30% and 0% of the articles. We call these sets DROP70, DROP30 and DROP0. There are about 1300 *a*'s and 2800 *the*'s in the section.

4.4 Identifying Noun Phrases

We would like to measure the degree to which the missing articles corrupted the parser output. We analyzed the following for each sentence: whether the correct NP heads were extracted; and, if so, whether the boundaries of the NPs were correct. DROP30 and DROP70 were POS-tagged and parsed, and then compared against DROP0.

97.6% of the sentences in DROP30 had all their NP heads correctly extracted. Among these sentences, 98.7% of the NPs had correct boundaries.

The accuracy rate for NP heads decreased to 94.7% for DROP70. Among the sentences in DROP70 with correct heads, 97.5% of the NPs had correctly boundaries.

We now turn our attention to how these errors affected performance in article generation.

4.5 Article Generation

We trained the log-linear model with TRAINGEN, TRAINGEN_{Minnen} and TRAINGEN_{base}, then performed the article generation task on all test sets. Table 1 shows the accuracy rates.

Our baseline accuracy rate on DROP0, 80.1%, is close to the corresponding rate (80.8% for the "head+its part-of-speech"feature) reported in (Minnen et al., 2000). Our best result, 87.7%, is an improvement over both (Minnen et al., 2000) and (Knight and Chander, 1994).

We added 8 more features (see §3.1) to TRAINGEN_{Minnen} to make up TRAINGEN. After adding the features **Words before/after head** and **POS of words before/after head**, the accuracy increased by more than 4%. In fact, these features dominated the 10 heaviest weights in our training; they were not used in (Minnen et al., 2000).

The **Words before/after NP** features gave another 0.8% boost to the accuracy. These features were also used in (Knight and Chander, 1994) but not in (Minnen et al., 2000). The **Hypernyms** feature, which placed NP heads under the WordNet semantic hierarchy, was intended to give a smoothing effect. It further raised the accuracy by 0.3%.

At this point, the biggest source of error was generating *null* instead of the correct *the*. We introduced the **Referent** feature to attack this problem. It had, however, only a modest effect. Among weights that involved this feature, the one with the largest magnitude was *(Article* = a) & (Referent = yes)*, at a meagre -0.71. The others were within ± 0.3. Table 2 is the final contingency table for TRAINGEN on DROP0.

The confusion between *null* and *the* remained the biggest challenge. The 656 misclassifications seemed rather heterogeneous. There was an almost even split between singular and plural NP heads; more than three quarters of these heads appeared in the list three times or less. The most frequent ones were <*number*> (22 times), *bond*, *year*, *security*, *court* (8 times), *fifth* and *show* (7 times).

As expected, the performance of TRAINGEN degraded on DROP30 and DROP70.

4.6 Article Restoration

So far, our experiments have not made use of the **Article** feature; articles in the original sentences are simply ignored. In the article restoration task, it is possible to take advantage of this feature.

We trained the log-linear model with TRAINDROP30, TRAINDROP70 and TRAINGEN. Our baseline was keeping the original sentences intact. The test sets were processed as follows: If an NP contained an article, the new article (that is, the output of the article generator) would replace it; otherwise, the new article would be inserted at the beginning of the NP. The final sentences were evaluated against the original sentences for three kinds of errors:

Deletions The number of articles deleted.

Substitutions The number of *a*'s replaced by *the*'s, and vice versa.

Insertions The number of articles inserted.

Training Set	DROP0	DROP30	DROP70
BASELINE	**0%**	30.3%	69.0%
TRAINDROP30	4.4%	**20.5%**	40.7%
TRAINDROP70	8.9%	22.3%	**38.5%**
TRAINGEN	43.0%	46.0%	49.4%

Table 3: Article error rate

The **article error rate** is the total number of errors divided by the number of articles in the original sentences.

The results in Table 3 reflect the intuition that, for a test set where n% of the articles have been dropped, the optimal model is the one that has been trained on sentences with n% of the articles missing. More generally, one could expect that the optimal training set is the one whose underlying confusion matrix is the most similar to that of the test set.

Whereas TRAINGEN ignores the original articles, both TRAINDROP30 and TRAINDROP70 led the model to become extremely conservative in deleting articles, and in changing *the* to *a*, or vice versa. Thus, the only major distinguishing characteristic between them was their aggressiveness in inserting articles: TRAINDROP70 was more aggressive than TRAINDROP30. Tables 4 to 6 illustrate the breakdown of the kinds of error contributing to the article error rate:

Training Set	DROP0	DROP30	DROP70
BASELINE	0%	30.3%	69.0%
TRAINDROP30	0.4%	13.0%	28.4%
TRAINDROP70	0.3%	9.7%	20.2%
TRAINGEN	19.3%	21.7%	23.9%

Table 4: Deletion error rate

The trends in the deletion error rate (Table 4) were quite straightforward: the rate was lower when the model inserted more articles, and when fewer articles were dropped in the original sentences.

Training Set	DROP0	DROP30	DROP70
BASELINE	0%	0%	0%
TRAINDROP30	0.0%	2.7%	6.5%
TRAINDROP70	0.0%	3.0%	7.1%
TRAINGEN	11.8%	11.3%	10.9%

Table 5: Substitution error rate

Most of the substitution errors (Table 5) were caused by the following: an article (e.g., *a*) was replaced by *null* in the test set; then, the wrong article (e.g., *the*) was generated to replace the *null*. In general, the substitution rate was higher when the model inserted more articles, and when more articles were dropped in the original sentences.

Training Set	DROP0	DROP30	DROP70
BASELINE	0%	0%	0%
TRAINDROP30	4.0%	4.9%	5.9%
TRAINDROP70	8.6%	9.7%	11.2%
TRAINGEN	11.9%	13.0%	14.6%

Table 6: Insertion error rate

The more aggressive the model was in inserting articles, the more likely it "over-inserted", pushing up the insertion error rate (Table 6). With the aggressiveness kept constant, it might not be obvious why the rate should rise as more articles were dropped in the test set. It turned out that, in many cases, inaccurate parsing (see §4.4) led to incorrect NP boundaries, and hence incorrect insertion points for articles.

As the wide range of error rates suggest, it is important to choose the optimal training set with respect to the input sentences. As one becomes more aggressive in inserting articles, the decreasing deletion rate is counter-balanced by the increasing substitution and insertion rates. How could one determine the optimal point?

Table 7 shows the changes in the number of articles, as a percentage of the number of articles in the final sentences. When running TRAINGEN on DROP30 and DROP70, there was an increase of 23.8% and 65.9% in the number of articles. These rates of increase were close to those obtained (24.4% and 66.0%) when running their respective optimal sets, TRAINDROP30 and TRAINDROP70. It appeared that TRAINGEN was able to provide a reasonable estimate of the number of articles that "should" be restored. When given new input sentences, one could use TRAINGEN to estimate the percentage of missing articles, then choose the most appropriate training set accordingly.

5 Future Work

5.1 Article Generation

We would like to improve the performance of the article generator. Our largest source of error is the confusion between *null* and *the*. In this work, we used predominantly intra-sentential features to disambiguate the articles. Article generation, however, clearly depends on previous sentences. Our only inter-sentential feature, **Refer-**

Training Set	DROP0	DROP30	DROP70
BASELINE	**0%**	0%	0%
TRAINDROP30	+3.9%	**+24.4%**	+60.1%
TRAINDROP70	+8.1%	+38.1%	**+66.0%**
TRAINGEN	-7.5%	+23.8%	+65.9%

Table 7: Change in the number of articles

ent, rather naïvely assumed that the referent was explicitly mentioned using the same noun within 5 preceding sentences. Techniques in anaphora resolution could help refine this feature.

5.2 Parser Robustness

The performance of the article generator degraded by more than 5% on when 30% of the articles in a sentence were dropped, and by more than 11% when 70% were dropped (see §4.5). This degradation was due to errors in the extraction of context features, and in identifying the NPs (see §4.4).

These errors could be reduced by retraining the POS tagger and the natural language parser on sentences with missing articles. New training sets for the tagger and parser could be readily created by dropping the article leaves from the Penn Treebank.

5.3 Weight Estimation

We used different confusion matrices to create training sets that simulated discrete percentages of dropped articles. Given some input sentences, the best one could do is to estimate their underlying confusion matrix, and choose the training set whose underlying matrix is the most similar.

Suppose a sentence is estimated to have half of its articles missing, but we do not have weights for a TRAIN-DROP50 set. Rather than retraining such a set from scratch, could we interpolate optimal weights for this sentence from existing weights?

5.4 Other Types of Grammatical Mistakes and Texts

We would like to lift our restrictions on the confusion matrix; in other words, to expand our task from restoring articles to correcting articles.

We have also identified a few other common categories of grammatical mistakes, such as the number of the NP head (singular vs. plural), and the verb tenses (present vs. past vs. continuous). For native speakers of languages that do not inflect nouns and verbs, it is a common mistake to use the root forms of nouns and verbs instead of the inflected form.

Finally, we would like to investigate how well the rules learned by our model generalize to other genres of texts. After all, most non-native speakers of English do not write in the style of the Wall Street Journal! We plan to train and test our model on other corpora and, if possible, on writing samples of non-native speakers.

6 Conclusion

We applied the log-linear model on the article generation task, using features drawn from a statistical natural language parser and WordNet. The feature set was progressively enriched with information from both inside and outside the NP, semantics, and discourse context. The final feature set yielded very competitive results.

We applied the same model to tackle the article restoration task, where sentences may have missing articles. On the one hand, article generation performance degraded significantly due to context extraction errors; this points to the need to adapt the tagger and parser to ungrammatical sentences. On the other hand, the articles that were already present in the sentence provided strong hints about the correct article; this points to the need for better methods for estimating the underlying confusion matrix of a sentence.

7 Acknowledgements

The author would like to thank Michael Collins and the four anonymous reviewers for their very helpful comments. This work is in part supported by a fellowship from the National Sciences and Engineering Research Council of Canada, and by the NTT Corporation.

References

Michael Collins. 1999. *Head-Driven Statistical Models for Natural Language Parsing*, Ph.D. Thesis, University of Pennsylvania, Philadelphia, PA.

Julia E. Heine. 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 (COLING/ACL-98)*, pages 519-525, Montréal, Canada.

Kevin Knight and Ishwar Chander. 1994. *Automated Postediting of Documents*, in *Proceedings of the 12th National Conference on Artificial Intelligence (AAAI-94)*, pages 779-784, Seattle, WA.

Guido Minnen, Francis Bond and Ann Copestake. 2000. *Memory-based Learning for Article Generation*, in *Proceedings of the 4th Conference on Computational Language Learning and the 2nd Learning Language in Logic Workshop (CoNLL/LLL-2000)*, pages 43-48, Lisbon, Portugal.

Adwait Ratnaparkhi. 1998. *Maximum Entropy Models for Natural Language Ambiguity Resolution* Ph.D. Thesis, University of Pennsylvania, Philadelphia, PA.