PFCG and Segmenter for Grammar Parsing Japanese Text

Josiah Putman
Josiah.K.Putman.20@Dartmouth.edu

Abstract

This project uses maximum likelihood probability estimates to train a PCFG language model for parsing pre-segmented Japanese text. The model was trained on and evaluated against the Keyaki Treebank. To create a complete end-to-end parser, a simple segmenter is implemented that inserts spaces between words in continuous Japanese text.

1 Introduction

Previous methodology for parsing Japanese text has generally used various grammar models, either hand-crafted or trained using supervised learning. In 2003, Masuichi et al. used Lexical-Functional Grammar (LFG) on pre-segmented text which had already been romanized into *romaji*, reaching 97% coverage on their test corpus using their custom grammar (Masuichi et al., 2003). As probabilistic and statistic models have gained popularity, statistical and machine learning models have become more common. Fang et al. in 2014 used PCFG with slight modifications to perform supervised parsing on the Keyaki Treebank, learning rules from human parsed data. The team chose to use heavy rule modifications, starting with a conversion to Chomsky Normal Form and various other manipulations to create efficient and accurate parses using the CYK algorithm (Fang et al., 2014).

The first portion of this project seeks to create a simplified version of Fang et al.'s experiment using Earley parsing and avoiding the complex modifications to the Keyaki grammar that were required for their performance. The second portion examines the task of segmentation, using a bi-gram model trained on pre-segmented text (also from the Keyaki Treebank).

2 NLP of the Japanese Language

Japanese is a particularly difficult language for computers to process due to its complex and ambiguous orthography, unique grammatical structure, and lack of spaces.

Written Japanese has three distinct writing systems: two syllabic alphabets, *hiragana* (ひらがな), *katakana* ((カタカナ), and one logogrammatic character system based on classical Chinese called *kanji* ((漢字). *Hiragana* is primarily used for grammar particles, verb conjugations, and common expressions. *Katakana* is generally used for loan words and foreign proper nouns, and *kanji* are often used for standard nouns, verb stems, etc. Occasionally, a fourth writing system called *romaji* is used, which is simply a romanization of Japanese phonetics using the Latin alphabet (Bond and Baldwin, 2016).

Due to the complexity of Japanese orthography, any given word may be represented several different ways, so NLP projects need to account for this using some form of normalization (such as working only in *romaji* as done by Masuichi).

On top of the challenges presented by the writing system, the lack of spaces also means that text must be segmented before being used to train any type of model. Segmentation methods range from rule-based approaches taking advantage of common boundary characters and *kana* to machine learning methods using LSTM and RNN models (Kitagawa and Komachi, 2017). Because it has been relatively unexplored in the literature, I attempted to use a simple bi-grams model with a custom segmenting algorithm to segment Japanese text.

Finally, Japanese adds the additional challenge of commonly omitted subjects and grammar particles. This allows sentences to be grammatically correct while leaving out key structural components. The Keyaki Treebank handles this by inserting ”null” objects into the tree to maintain the structure. For the purposes of parsing, most algorithms simply ignore these null characters, which is how my PCFG implementation handles it.

3 Keyaki Treebank

For both the PCFG and the segmenter, the Keyaki Treebank was used as the training and test corpus. The Keyaki Treebank contains 217 parse trees from blog posts, 484 trees of Japanese Law, 1600 from newspapers, 1177 from telephone calls, 7733 from textbooks and 2464 from Wikipedia for a total of 13675 trees (Butler et al., 2012).

For this project, I used the Aozora texts and the telephone call transcriptions.

4 PCFG

A simple PCFG was implemented using Python 3. The training process involves counting symbol and rule occurrences in the treebank.

4.1 Probability calculations

The maximum likelihood can be calculated using:

$$qML(\alpha \rightarrow \beta) = \frac{\text{Count}(\alpha \rightarrow \beta)}{\text{Count}(\alpha)}$$

where Count$(\alpha \rightarrow \beta)$ is the number of occurrences of the rule $\alpha \rightarrow \beta$ in T_{train} and Count(α) is the number of occurrences of non-terminal α in T_{train}.

4.2 Parsing using Earley Parsing

Earley Parsing is performed using the lark-parser Python package. For this project, I have developed a custom adapter that parses Keyaki Treebank parse trees, trains a PCFG, and outputs the rules of the created grammar in the ‘.lark‘ file format. Unfortunately the Lark Parser seems to have some difficulty with certain Japanese symbols, as it fails on specific trees from the tree-bank.

4.3 Disambiguation

Earley parsing using Lark provides the option for explicit ambiguity, i.e. representing ambiguity in the resulting parse tree.

My implemented PCFG chooses the most probable parse by recursively calculating the probability of the subtree at each ambiguous point and choosing the most probable. This bottom-up approach is much more efficient than generating every possible parse tree form the ambiguous tree and then calculating top-down probabilities of each tree.

The probability of each tree t containing rules $\alpha_1 \rightarrow \beta_1, \ldots, \alpha_n \rightarrow \beta_n$ is calculated using the following formula:

$$p(t) = \prod i = 1q(\alpha_i \rightarrow \beta_i)$$

(Collins, 2010)

To prevent floating point underflow, instead of taking the product of direct probabilities, I take the sum of logarithmic probabilities.

5 Segmenter

The following algorithm explains the segmentation process. S represents an ordered set of possible segmentation indices.

```
Data: String $text$
$S \leftarrow [[]]$;
for $i = 0;\ i < text.size();\ i++$ do
    $S_{new}$ =copy($S$);
    for $j = 0;\ j < S.size();\ j++$ do
        $x \leftarrow text[S[j][-1], i]$;
        if $x \in V$ then
            if $x \in P$ then
                $S$.append(copy($S[j]$));
            $S[j]$.append($i$);
        else
            if $x \notin P$ then
                $S[j] = []$;
    end
    $S = \{s \in S_{new} | s \neq []\}$;
    return $\arg\max_{s \in S} p(s)$;
end
```

Algorithm 1: Segmentation algorithm

6 Analysis

In this section, the performance of the PCFG, segmenter, and combined system are analyzed and compared to previous results.

The following table shows the results of the PCFG and Segmenter when tested on subsets of the Keyaki Treebank corpus. In the case of the PCFG, accuracy represents the proportion of correctly tagged terminals versus the total number of terminals in a given string. For the segmenter, accuracy represents the total number of matching segments versus the total number of tokens in the true segmentation.

6.1 PCFG Performance

Precision and recall are calculated using PARSEVAL definitions as follows:

$$recall = \frac{\text{\#correct constituents in hyp. tree}}{\text{total constituents in ref. tree}}$$

$$precision = \frac{\text{\#correct constituents in hyp. tree}}{\text{total constituents in hyp. tree}}$$

Table 1 shows the results of running the PCFG on 50 random sentences extracted from the training corpus (tested sentences are omitted from training).

Corpus	**Precision**	**Recall**	**F-Score**
Aozora	0.876	0.927	0.899
Spoken	0.931	0.996	0.959

Table 1: Performance analysis.

Fang et al., who also used PCFG and various enhancements, received the following results shown in 2., calculated using PARSEVAL (Fang et al., 2014).

Model	**precision**	**recall**	F-score
Vanilla	0.657	0.679	0.668
Enhanced	0.799	0.806	0.803

Table 2: Fang's Performance analysis

Although the results of this project are better than Fang et al., my PCFG was only able to work on a small subset of the Keyaki corpus due to problems with the Lark parsing library, so the comparison is not completely fair. Fang et al.'s solution is much more robust and applies to the entire corpus.

6.2 Segmenter Performance

Precision and recall are calculated based on the definitions provided by Palmer et al., where recall is the percentage of tokens from the true parse that are present in the segmenter's parse and precision is the percentage of tokens in the segmentation that are in the same position as that in the true parse (Palmer, 1997).

Table 3 shows the results of running the segmenter on 50 random sentences from the training corpus (tested sentences are omitted from training).

Corpus	**Precision**	**Recall**	**F-Score**
Aozora	1.0	1.0	1.0
Spoken	1.0	1.0	1.0

Table 3: Performance analysis.

6.3 Combined performance

Because the segmenter adds no error to the system, the PCFG works just as well with the segmenter used as a pre-processing step and allows the program to function as an end-to-end grammar parser.

7 Limitations and Future Work

Because the PCFG does not consider context, it may not be a perfect representation of the grammar of Japanese. PCFGs also make independence assumptions with regards to probabilities that are not necessarily safe to make in the case of Japanese grammar. Instead of a grammar model, perhaps more adaptive deep learning algorithms could be applied to generate parse trees.

According to the Lark library website, the Earley parser is severely inefficient compared to their other parser implementations such as CYK or LARL(1). In the future, it would definitely be worthwhile to convert the grammar to Chomsky Normal Form and run those more efficient parsers. One drawback is that these parsers are unable to handle ambiguity, so unless there is an effective method of removing ambiguity from a grammar as complex as the one learned from the Keyaki Treebank, then these other parsers may not be feasible.

Lark was also unable to handle certain aspects of the grammar, but was unhelpful with its error messages. Instead of using this parsing library, writing a custom parser that performs a modified version of Earley's algorithm would be a nice extension of this work.

With regards to the segmenter, several of the speed-memory trade-offs of the algorithm should be considered. Although keeping track of all possible prefixes makes the segmentation process much faster, it increases the memory cost exponentially with respect to the size of the vocabulary. In the case of Japanese, rule-based segmenters such as the TinySegmenter perform nearly as well but with much smaller memory costs.

This segmenter, being based on an N-grams model, is not very flexible when used on texts that have unseen vocabulary. Even with Laplace Smoothing applied, the algorithm has no sense of where unseen words should be separated, whereas rule-based or more advanced ML methods could handle unseen input much more precisely.

References

[Bond and Baldwin2016] Francis Bond and Timothy Baldwin. 2016. Introduction to japanese computational linguistics.

[Butler et al.2012] Alastair Butler, Zhu Hong, Tomoko Hotta, Ruriko Otomo, Kei Yoshimoto, and Zhen Zhou. 2012. Keyaki treebank: phrase structure with functional information for japanese. In *Proceedings of Text Annotation Workshop*.

[Collins2010] Michael Collins. 2010. Probabilistic context-free grammars (pcfgs). 2013. *URL http://www. cs. columbia. edu/˜ mcollins/courses/nlp2011/notes/pcfgs. pdf.*

[Fang et al.2014] Tsaiwei Fang, Alastair Butler, and Kei Yoshimoto. 2014. Parsing japanese with a pcfg treebank grammar. In *Proceedings of the Twentieth Annual Meeting of the Association of Natural Language Processing*, pages 432–435.

[Horn and Butler2016] Stephen Wright Horn and Alastair Butler. 2016. Npcmj interface and users manual no. 1 (2018-11-30).

[Kitagawa and Komachi2017] Yoshiaki Kitagawa and Mamoru Komachi. 2017. Long short-term memory for japanese word segmentation. *arXiv preprint arXiv:1709.08011*.

[Masuichi et al.2003] Hiroshi Masuichi, Tomoko Okuma, Hiroki Yoshimura, and Yasunari Harada. 2003. Japanese parser on the basis of the lexical-functional grammar formalism and its evaluation. In *Proceedings of The 17th Pacific Asia Conference on Language, Information and Computation*, pages 298–309.

[Palmer1997] David D Palmer. 1997. A trainable rule-based algorithm for word segmentation. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics and Eighth Conference of the European Chapter of the Association for Computational Linguistics*, pages 321–328. Association for Computational Linguistics.

A Keyaki Treebank Reference

QUOT	quote
-LRB-	left bracket
-RRB-	right bracket
PU	punctuation
ADJI	i-adjective
ADJN	na-adjective
ADV	adverb
AX	auxiliary verb (including copula)
AXD	auxiliary verb, past tense
CL	classifier
CONJ	coordinating conjunction
D	determiner
FW	foreign word
INTJ	interjection
MD	modal element
N	noun
NEG	negation
NPR	proper noun
NUM	numeral
P	particle
PASS	passive
PNL	prenominal
PRO	pronoun
Q	quantifier
QN	noun with quantifier
SYM	symbol
VB	verb (or verb stem)
VB0	light verb
VB2	secondary verb
WADV	indeterminate adverb
WD	indeterminate determiner
WNUM	indeterminate numeral
WPRO	indeterminate pronoun

Table 4: Part-of-speech tags

(Horn and Butler, 2016)

ADVP	adverb phrase
CONJP	conjunction phrase
CP-EXL	exclamative
CP-FINAL	projection for sentence final particle
CP-QUE	question (direct or indirect)
CP-THT	complementizer clause
FRAG	fragment
FRAG-IMP	imperative with fragment
FS	false start
INTJP	interjection phrase
IP-ADV	adverbial clause
IP-EMB	gapless noun-modifying clause
IP-IMP	imperative clause
IP-SMC	small clause
IP-MAT	matrix clause
IP-REL	relative clause
IP-SUB	clause under CP* layer
IP-NML	nominalized clause
multi-sentence	multiple sentence
NML	intermediate nominal layer
NP	noun phrase
NP-ADV	adverbial noun phrase
NP-LGS	logical subject noun phrase
NP-LOC	locational noun phrase
NP-MSR	measure noun phrase
NP-OB1	noun phrase first object
NP-OB2	noun phrase second object
NP-PRD	predicate noun phrase
NP-SBJ	noun phrase subject
NP-SBJ2	noun phrase second subject
NP-TMP	temporal noun phrase
NP-TPC	topic noun phrase
NP-VOC	vocative noun phrase
NUMCLP	numeral-classifier phrase
PNLP	prenominal phrase
PP	particle phrase
PP-PRP	purpositive particle phrase
PRN	parenthetical

Table 5: Syntactic tags

CND	conditional
SCON	subordinate conjunction
CONJ	coordinate conjunction
CODE	code
LS	list item
LST	list
META	meta information

Table 6: Other tags